Automated Map Generation Process for Tiled Raster Maps

Aalto University
School of Engineering
Department of Real Estate, Planning and Geoinformatics
Master's Thesis

Espoo 30.11.2015

B.Sc. (Tech) Christian Koski

Supervisor: Professor Kirsi Virrantaus
Instructor: D.Sc. (Tech) Pyry Kettunen

Aalto University
School of Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi

Abstract of master's thesis

Author Christian Koski

Title of thesis Automated Map Generation Process for Tiled Raster Maps

Degree programme Degree Programme in Geomatics

Major Geoinformatics **Code** M3002

Thesis supervisor Professor Kirsi Virrantaus

Thesis advisor(s) D.Sc. (Tech) Pyry Kettunen

Date 30.11.2015 **Number of pages** 72 **Language** English

Abstract

Free and open source software (FOSS) for web mapping has progressed rapidly in the past decade, providing a variety of tools for tiled map making. Tiled maps increase the speed at which map clients are able to fill the map view for users, enhancing the user experience. In this thesis the development of an automated map generation process for three varying types of multi-scale tiled raster maps, covering the area of Finland, is presented. The aim of the study was to provide insight on tiled map making with FOSS, answering questions such as; how the process can be built from existing FOSS products; what decisions are required to be made and what challenges are encountered during the development of the process; and, how can tiles from different areas of maps be rendered simultaneously without affecting the final outcome. Rendering tiles from different areas simultaneously with several hardware instances decreases the overall rendering time.

The theoretical and technical background of tiled map making was first reviewed. The development of the map generation process was then planned and the source data was reviewed. In the first stage of the development, a system that renders tiles, but does not include any map type specific elements, was built. In the second stage of the development, the map type specific map production flow lines were added to the map generation process, including data processing operations and stylesheets. In the third and final stage of the development the process was qualitatively evaluated.

The finished map generation process first transforms the source data to a form where it can be used directly for rendering maps with multiple data processing operations. The tiles are then rendered, and each tile is stored as a separate file in a folder structure. The development of the process revealed that building data processing operations take up the majority of the time that is used for the development. The slowest parts of the finished map generation process is transforming digital elevation models to hillshading and contour lines, and rendering the tiles. These two parts of the process were divided geographically into forty areas that can be processed separately without affecting the maps that are generated. The process should be developed further to generate hillshading, contour lines, and tiles faster, for example, by moving these processes to multiple hardware instances.

Keywords tiled maps, map generation, map making, free and open source software

Aalto-yliopisto
Insinööritieteiden
korkeakoulu

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi

Diplomityön tiivistelmä

Tekijä Christian Koski

Työn nimi Prosessi tiilitettyjen rasterikarttojen automaattiseen tuottamiseen

Koulutusohjelma Geomatiikka

Pääaine Geoinformatiikka **Koodi** M3002

Työn valvoja Professori Kirsi Virrantaus

Työn ohjaaja(t) TkT Pyry Kettunen

Päivämäärä 30.11.2015 **Sivumäärä** 72 **Kieli** Englanti

Tiivistelmä

Web-karttojen valmistamiseen kehitetyt vapaat, avoimen lähdekoodin ohjelmistot ovat kehittyneet nopeasti viimeisen vuosikymmenen aikana, sisältäen työkaluja tiilitettyjen karttojen valmistamiseen. Tiilitetyt kartat tehostavat karttakuvapalveluiden suorituskykyä, koska karttojen osa-alueita voidaan noutaa ja esittää näytöllä erikseen. Tässä diplomityössä esitellään automatisoidun tiilitettyjen rasterikarttojen tuottamisprosessin kehittäminen. Prosessi tuottaa automaattisesti kolme erityyppistä monimittakaavaista koko Suomen kattavaa rasterikarttaa. Työn tavoitteena oli tarjota näkemystä tiilitettyjen karttojen valmistamiseen vapailla, avoimen lähdekoodin ohjelmistoilla ja vastata kysymyksiin: mitä päätöksiä ja haasteita kohdataan prosessin kehityksessä, mitkä ovat kehityksen haastavimmat vaiheet ja kuinka samaan karttaan kuuluvia tiiliä voidaan renderöidä samanaikaisesti useammalta eri alueelta vaikuttamatta lopulliseen karttatuotteeseen. Renderöimällä alueita samanaikaisesti, voidaan tiilten renderöinti jakaa monelle eri tietokoneelle, vähentäen renderöintiin kuluvaa kokonaisaikaa.

Rasterikarttojen tuotantoprosessin kehittämiseksi käytiin ensiksi läpi teoreettinen ja tekninen tausta jonka pohjalle prosessin kehittäminen rakentuu. Tämän jälkeen suunniteltiin prosessin kehittäminen ja käytiin läpi lähdeaineistot. Kehitystyö aloitettiin rakentamalla järjestelmä joka renderöi tiiliä, mutta joka ei sisällä yksittäisiin karttoihin liittyvää aineistojen prosessointia tai tyylitiedostoja. Kehityksen toisessa vaiheessa karttoihin liittyvät elementit lisättiin karttatuotantoprosessiin. Kehityksen kolmannessa ja viimeisessä vaiheessa karttatuotantoprosessi arvioitiin kvalitatiivisesti.

Valmistunut karttatuotantoprosessi muuntaa ensin useammassa erillisessä prosessissa lähdeaineistot muotoon jossa niitä pystytään suoraan käyttämään karttojen renderöintiin. Tämän jälkeen tiilet renderöidään ja jokainen tiili tallennetaan erillisenä tiedostona. Prosessin kehitystyö paljasti, että lähtöaineiston prosessointitehtävien kehittäminen vie suurimman osan kehitysajasta ja lähdeaineistojen prosessointivaiheiden kehitys on kehitysvaiheen haastavin osa. Valmistuneen karttatuotantoprosessin hitaimmat vaiheet ovat vinovalovarjosteiden ja korkeuskäyrien tuottaminen korkeusmalleista sekä tiilien renderöinti. Kehitetyssä prosessissa nämä vaiheet jaetaan maantieteellisesti neljäänkymmeneen osaan, jotka käsitellään erikseen. Jako ei vaikuta lopullisiin karttatuotteisiin. Prosessia tulee kehittää nopeammaksi esimerkiksi viemällä eri alueiden korkeusmallien prosessointi ja tiilien tuottaminen useammalle eri tietokoneelle, joilla ne prosessoidaan samanaikaisesti nopeuttaen prosessointiin kuluvaa kokonaisaikaa.

Avainsanat kartanvalmistus, tiilitetty kartta, kartan tuottaminen, vapaa avoimen lähdekoodin ohjelmisto

Acknowledgements

This thesis was made at the Finnish Geospatial Research Institution (FGI), Department of Geoinformatics and Cartography. Funding from the Academy of Finland (grants 259557 and 292753) is gratefully acknowledged.

I express my sincere gratitude to my instructor, Dr. Pyry Kettunen for all the valuable advice, insight and guidance I have received that has greatly improved this thesis. I thank my supervisor, Prof. Kirsi Virrantaus who gave support and advice during the process of finding a suitable thesis subject. I thank Prof. Tapani Sarjakoski for giving me the opportunity to make my Master's thesis on an interesting subject at the FGI, it has been a truly great experience. I acknowledge with much appreciation Dr. Juha Oksanen, Cecilia Bergman, and Eero Hietanen for providing me with their expertise, help, and support, and thank all of my colleagues at the FGI for a great work atmosphere. I am also grateful to Dr. Paula Ahonen-Rainio for helping me with the final stage of this thesis, and for having been a great lecturer and teacher during my studies at the Aalto University. Finally, I thank my family and friends for supporting me throughout my studies.

Kirkkonummi, 30.11.2015

Christian Koski

Contents

Abbreviations

AJAX	Asynchronous JavaScript and XML
CRS	Coordinate Reference System
DBMS	Database Management System
DCM	Digital Cartographic Model
DEM	Digital Elevation Model
DLM	Digital Landscape Model
GML	Geographic Markup Language
FGI	Finnish Geospatial Research Institute
FOSS	Free and Open Source Software
FOSS4G	Free and Open Source Software for GIS
GDAL	Geospatial Data Abstraction Library
GIMP	GNU Image Manipulation Program
GUI	Graphical User Interface
JHS	Public Administraation Recommendations
JPEG	Joint Photographic Experts Group
NLS	National Land Survey of Finland
ODBMS	Object Database Management System
OGC	Open Geospatial Consortium
ORDBMS	Object Relational Database Management System
PNG	Portable Network Graphics
RDBMS	Relational Database Management System
SQL	Structured Query Language
WMTS	Web Map Tile Service
XML	Extensible Markup Language

List of Figures and Tables

1 Introduction

1.1 Background

Web mapping techniques have progressed rapidly since 2005, the year Google first introduced Google Maps, credited for being the first web map application to combine tiled maps with Asynchronous JavaScript and XML (AJAX) (Tsou, 2011). Tiled maps reduce the amount of data that is required to be loaded and displayed on screen by map clients, as map clients can ignore tiles that do not contribute to filling the map view (Sample & Ioup, 2010).

The Finnish Geodetic Institute (today the Finnish Geospatial Research Institute [FGI] in the National Land Survey of Finland [NLS]) produced a tiled map series of the Nuuksio National Park area as part of two projects, MenoMaps and MenoMaps II that were carried out between 2008 and 2013 (Oksanen, et al., 2011). The maps were made for being published on multiple platforms, including mobile phones, printed maps, web-browsers, and a multi-touch screen (Oksanen, et al., 2011; Kettunen, et al., 2012). The MenoMaps map series consists of five multi-scale tiled maps with different use context (Oksanen, et al., 2011). The map types were called, 'Topographic map', 'Forest map', 'Winter map', 'Relief map' and 'Orthophoto map' (Oksanen, et al., 2011). The maps were made with a three stage process, consisting of a data processing stage, a map design stage and an image processing stage. Tools that were used in the making of the maps included Terrasolid TerraScan and TerraPhoto, ArcGIS 9.3.1 desktop, and Adobe Photoshop CS4.

In a following project of the Finnish Geodetic Institute, five multi-scale tiled raster maps were produced that covered the Viherkehä area in southern Finland, expanding the area that was covered by the MenoMaps map series. Four of the five maps were based on map types from the MenoMaps map series (‘Topographic map’, ‘Forest Map’, ‘Relief map’ and ‘Orthophoto map’). In addition, the Viherkehä maps included a map of a fifth type, the 'Superficial Deposits map'. The maps were produced with a similar map making process to the one used in MenoMaps, utilizing ArcGIS and Adobe Photoshop.

Following the making of the Viherkehä maps, the FGI desired to study the making of multi-scale tiled raster maps that cover the whole area of Finland. Initially three maps were to be made. The map types were to be based on the ‘Topographic map’, ‘Forest map’ and ‘Superficial Deposits map’ map types from the Viherkehä maps.

The source data volume used for the new maps was going to be significantly larger than that used for the MenoMaps map series and the Viherkehä maps. Braga et al. (2014) state that when data volume increases in GIS applications, it is critical to ensure system performance and scalability. It became evident that, because the processes that were used to produce the MenoMaps map series and the Viherkehä maps required a significant amount of manual work through graphical user interfaces (GUIs), these processes would not suite well for producing the new maps. Manual work is time consuming, and GUIs are likely to become overburdened by high resolution datasets of large areas.

The new map making process was to include automated data processing and tile rendering. Some of the source datasets that were used in the MenoMaps map series and Viherkehä maps were not available of the whole area of Finland. New source data was needed that required the map making process to be extended with new data processing operations. As a new approach, the use of free and open source software (FOSS) for building the process was encouraged. Multi-scale tiled maps may require millions of tiles to be rendered. To operate efficiently, tile rendering may be required to be divided onto multiple hardware instances (Sample & Ioup, 2010). Tile rendering in the new process was to be designed so that it would be possible in the future development work to divide the rendering of maps onto multiple hardware instances.

Map production is a multi-phase process that begins with deriving geographic data from real life objects, and ends with finished maps (Longley, et al., 2015). Today, geographic data is widely available for free, allowing organizations to create maps with much lighter processes. In this thesis, such processes are referred to as map making. Map making includes five phases: acquiring existing geographic data, processing source data so that it may be used for rendering, applying map design, rendering maps, and storing maps. Map making cannot be fully automated, because acquiring source data and applying map design typically require human interaction. However, once data has been acquired and map design has been applied, the remaining three phases can be processed in a single automated process. For the purpose of this thesis, we refer to such a process as a map generation process. A map generation process is thus a subprocess of map making, and map making is a subprocess of map production (*Figure 1*).

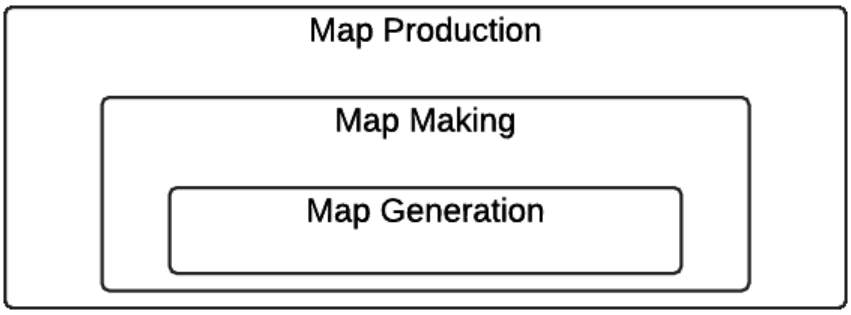

Figure 1. The relationship of Map Prodction, Map Making and Map Generation.

1.2 Aim of the Study and Research Questions

The aim of this thesis is to provide insight into the development of map generation processes for multi-scale tiled raster maps with national extent, using FOSS tools. Documentation on such processes is scarce. Although, private and public mapping organizations build and manage tiled raster map generation processes with FOSS tools, the processes are typically built for generating maps with a single use context (e.g. Sipilä, 2015).

The thesis aims to answer the following questions:

- How can a process, that generates multi-scale tiled raster maps with varying use contexts and with coverage of the area of Finland, be developed with FOSS tools?
- What are the most important decisions that are required to be made, and the major challenges that are faced in the development of such a process?
- How effective is processing of source data and rendering of maps in such a process, and how can parts of maps be rendered separately without it affecting the final outcome?

Rapid changes in software and hardware leads to quick expiration of information found on them in books (Tyner, 2010) and research papers (Tsou & Smith, 2011). Discovering the software with specific functionality requires recent analysis of software products that include the desired properties. Because software selection is an important part of the development of the map generation processes, this thesis also aims to present FOSS projects that were available for tiled map making during the time of the development. Software that was selected for the process, and their alternatives are presented and discussed in this thesis.

The study was carried out by first, reviewing the theoretical and technical background of tiled map making. Second, a map generation process, for producing three multi-scale tiled raster maps, with national extent, was developed, based on the 'Topograpic map', 'Forest map' and 'Superficial Deposits map' map types from the Viherkehä maps. Third, the map generation process was qualitatively evaluated by comparing its functionality and properties to the capabilities that were desired from it, and by assessing its data processing and rendering quality and speed.

1.3 Structure of the Thesis

This thesis is divided into seven chapters. In Chapter 2, the theoretical and technical background of tiled map making is presented, including map making in general, elements of tiled map making and software in tiled map making. In Chapter 3 the methods and materials for the development of the map generation process is presented, including the planning of the development process, the source data and the software selection methods that are used. In Chapter 4, the development of the tile rendering system, the core component of the map generation process, is presented. In Chapter 5 the adding of data processing and map design to the process is presented. In Chapter 6 the outcome of the development of the map generation process is presented and qualitatively evaluated, and findings from the development process are discussed. In Chapter 7 concluding remarks are presented.

2 Theoretical and Technical Background

This chapter presents the theoretical and technical background on which the development of the map generation process that is presented in this thesis is based. The chapter is divided into three sections. In Section 2.1, map making is presented in general, including the historical background of web maps, map production processes, map series and map design. In Section 2.2, tiled map making is presented, including, tiled maps, tiles, logical tile schemes, vector tiles and standards for tiled systems and maps. In Section 2.3, software for tiled map making is overviewed. The chapter concludes with a summary of these topics.

2.1 Map Making Processes

2.1.1 Historical Background of Web Maps

In the longest era of cartography, maps were made by hand (Niemelä, 2004; Robinson, et al., 1984), usinhandheldld tools like brushes and quills (Robinson, et al., 1984). The first major step towards automated map production was the development of techniques that enabled mass-production of maps (Niemelä, 2004; Robinson, et al., 1984). Mechanical technology increased the speed and efficiency of map making, resulting in more affordable and accessible maps for a larger number of users (Robinson, et al., 1984)

Before the 1990s maps were mainly published as printed maps. During the early days of the Internet, digital web maps began to emerge (Longley, et al., 2015). The introduction of the Xerox PARC Viewer in 1993 is credited as marking the beginning of web mapping (Haklay, et al., 2008; Longley, et al., 2015). Early web maps were simple and cumbersome, often consisting of a single image that was loaded from a server to a map client, with functionality restricted to panning the map with the arrow keys (Haklay, et al., 2008; Sample & Ioup, 2010). Each time the user would pan the map, the map client was required to request the server for a new map image (Haklay, et al., 2008). This period of static web maps lasted for more than a decade (Haklay, et al., 2008).

In 2005 Google presented a new web mapping technique by releasing Google Maps (Sample & Ioup, 2010; Tsou, 2011; Batty, et al., 2010), combining tiled maps with AJAX (Tsou, 2011). Pairing tiled maps with AJAX, improved the performance, and dramatically enhanced the user experience, of web maps (Tsou, 2005). Several other

widely used web maps quickly adopted the technique, including Bing Maps and Yahoo! Maps (Sample & Ioup, 2010). The introduction of Google Maps sparked a series of new innovations. In 2006, Google Maps API was released, which enabled anyone with skills in JavaScript to develop map mashups, combining the base map from Google Maps with data from other sources (Schmidt & Weiser, 2012). In mid-2007 there were 50 000 web map mashups based on Google Maps API (Haklay, et al., 2008). A wide variety of FOSS for developing web map applications began to emerge (Roth, et al., 2014). Web mapping technologies are still rapidly evolving, increasing in both flexibility and interoperability (Roth, et al., 2014). It is the fastest growing category of GIS software, both amongst commercial GIS software (Longley, et al., 2015) and FOSS GIS projects (Steiniger & Hunter, 2013).

During the digital era, map production processes have gradually become more automated. Today, most map production sub-processes are almost completely automated. However, there are still a number of technical and scientific issues, mostly concerning data generalization, which prevent the production processes to become fully automated. (Longley, et al., 2015)

2.1.2 Map Production Processes

Longley et al. (2015) define map production as the production of formal maps according to established cartographic conventions. These formal maps are regarded as being different from map visualizations, that refer to transitory maps and map-like visualizations that are used for displaying, analyzing, editing and querying geographic information (Longley, et al., 2015).

Typically, maps are divided into two main classes, topographic and thematic maps (Longley, et al., 2015). Other categorizations include division into general-purpose, special purpose and thematic maps (Tyner, 2010); or, general maps, thematic maps, and charts (Robinson, et al., 1984). The definition of map type may also be extended to differentiate maps of different scales (Virrantaus, et al., 2009; Robinson, et al., 1984), and paper and digital maps. (Virrantaus, et al., 2009). Map production processes vary depending on the map type (Virrantaus, et al., 2009).

Hardy et al. (2004) and Longley et al. (2015) present a view of modern map production processes that revolve around a single geographic database. The database is built using a

data model that represents elements of the real world. This data model is referred to as a digital landscape model (DLM) (Hardy, et al., 2004; Longley, et al., 2015). Objects in the DLM do not have a cartographic representation, and cannot as such be used directly for map making. A map production process essentially transforms this database into a formal map by first deriving cartographic representations of objects in the DLM database (Longley, et al., 2015). Hardy et al. (2004) and Longley et al. (2015) refer to the data model that represent derived objects as a digital cartographic model (DCM). A single map with a single DCM is just one of many possible products of a DLM database. Ideally a single continuous, scale-free DLM database could be used to automatically produce multiple different types of map products, resulting in multiple DCMs. However, scientific and technical issues make this difficult to achieve (Hardy, et al., 2004; Longley, et al., 2015)

2.1.3 Map Series

Map series are collections of maps that share common elements, for example symbology and projection (Longley, et al., 2015). Producing map series with a single map making process can save time and expenses, reducing duplicate efforts that can occur from several separate processes (Hardy, et al., 2004).

To produce map series in such a way, the base production process is extended with additional production flow lines, each resulting in a single finished map-product. In the map production model presented by Hardy et al. (2004) and Longley et al. (2015) each production flow line has its own DCM. However, it may not be possible to produce all these DCMs directly from a base DLM. Coarser scale DLMs are created first by generalizing the data model of the base DLM. Because of efficiency, and because all database generalization is not fully automated, creating these additional DLM databases may require manual work. Also, each separate production flow line will typically require additional work to automate and attach them to the main process. (Hardy, et al., 2004; Longley, et al., 2015)

2.1.4 Map Design

Map design has two meanings, referring both to layout and planning (Tyner, 2010). Layout, also called composition, refers to the creation and placement of map elements, for example, body, title, legend, and scale (Tyner, 2010; Longley, et al., 2015). On the

other hand, planning refers to decisions on what symbols, scale and projection to use (Tyner, 2010). Map design aims to "share information, highlight patterns and processes, and illustrate results" (Longley, et al., 2015, p. 246). Map design also aims at achieving aesthetically pleasing maps, although it is not the primary aim (Longley, et al., 2015). Map design is a core element of any map making process.

2.2 Tiled Map Making

Techniques for making tiled maps have progressed rapidly and become widely adopted in map making. The widespread use of tiled maps is largely credited to the high performance gain from loading only small parts of the maps on screen at a time, as map clients are only required to load enough tiles to cover the users map view (Tsou, 2005; Sample & Ioup, 2010).

2.2.1 Tiled Maps

A tiled map consists of a collection of image fragments called tiles. The tiles are organized into regularly spaced grids referred to as a tile matrix. Tile matrices have a resolution level that specifies the resolution of the tiles in it, for example the resolution level may be given in meters per pixel, corresponding to the width and height in meters that each pixel of each tile covers. Multi-scale tiled maps consist of several tile matrices of varying resolution levels. A set of tile matrices that makes a multi-scale map is referred to as a tile-matrix set. (OGC, 2010)

Tiled maps have unique aspects compared to other maps. Tiles are required to be produced according to a logical tile scheme that defines properties of the tile matrix set, including projection and the tile addressing scheme (Sample & Ioup, 2010). Tile storage is also a key issue in tiled map making (Sample & Ioup, 2010). Tiles may be rendered and stored in advance, or rendered on-demand, as the map client requests more tiles (Quinn & Gahegan, 2010). The popularity of tiled maps and the need to define properties for tiled systems has sparked initiatives to create standards for them (OGC, 2010).

The majority of tiled web maps today are raster maps (Antoniou, et al., 2009; Gaffuri, 2012; Dufilie & Grinstein, 2014), including maps in major services such as Google Maps and OpenStreetMap (Corcoran, et al., 2011). Recently techniques have been studied to enable the use of tiled vector maps in map clients (Dufilie & Grinstein, 2014).

2.2.2 Tiles

Tiles are the central elements of tiled maps. Typically, tiles are raster bitmap images, although vector tiles have started to emerge. Raster tiles are also sometimes referred to as image tiles (Sample & Ioup, 2010). Tiles are rectangular representations of geographical data that each have a unique address, marking their location in a tile matrix set (OGC, 2010).

A key property of image tiles is their image format. There are hundreds of image formats. However, only a few formats should be considered for image tiles (OGC, 2010; Sample & Ioup, 2010). Image format dictates the possible compression schemes, color depth, and the ability to create transparent tiles (Sample & Ioup, 2010). Map clients are also typically compatible with only certain image formats. Two image formats are typically recommended for tiled maps, Portable Network Graphics (PNG) and Joint Photographic Experts Group (JPEG) (OGC, 2010; Sample & Ioup, 2010). The main difference between these two are that PNG images are lossless, meaning that when compressing them, no information is lost. In addition, JPEG images do not support transparency (OGC, 2010; Sample & Ioup, 2010). On the other hand the file size of PNG images can be significantly larger than JPEG file sizes (Sample & Ioup, 2010).

Sample and Ioup (2010) state that choosing a size for tiles is one the most important decisions for the developer to make when making tiled maps. Tile sizes are typically given in pixels. Tiles can be of any size and the tile size may even vary for different resolution levels, or within a resolution level. However, using the same size for all tiles in a tile matrix set is beneficial, because map clients typically only support this type of sets. In addition, the mathematics is simpler for a system that uses tiles with the same width and height, and that use a tile scheme where each tile matrix quadruples the amount of tiles that were included in the previous smaller resolution tile matrix. Tiles that are small in size will require more power from the hardware, as there are more objects that need to be identified, fetched and loaded on screen. However, large tiles reduce the performance gain that is achieved from using tiles, because large tiles result in more redundant parts outside the map view.

Sample and Ioup (2010) compared the efficiency of loading tiles of different sizes to a map view. The results showed that the 128 x 128 pixel tiles were the most efficient. However, when the tile image files were compressed, the 512 x 512 sized tiles were the

most efficient. Sample and Ioup (2010) conclude that they recommend the use of tiles that are the size of 512 x 512 pixels. The OGC WMTS standard does not define a standard tile size (OGC, 2010). However, the standard includes several alternative tile schemes for tiled maps with a tile size of 256 x 256 pixels (OGC, 2010). The Finnish Public Administration Recommendation JHS 180 recommends a tile size of 256 x 256 pixels for maps of Finland (JUHTA, 2013).

In addition to the bitmap and tile address, tiles may include metadata. Relevant metadata is, for example, the date when the tiles were created and what source data were used. Depending on the tile storage approach, metadata could be stored either in separate files, in a large file for multiple tiles, or in a database table as additional attributes of tiles. (Sample & Ioup, 2010)

2.2.3 Logical Tile Schemes

Tiles do not contain direct information about their geographical position. Instead, map clients are provided with the coordinates of the top left corner of the extent of the tile matrix set (OGC, 2010; Sample & Ioup, 2010) In addition map clients require tiles to be created according to a logical tile scheme (Sample & Ioup, 2010) that enables them to identify, fetch, and display the correct tiles from a tile matrix set. The logical tile scheme defines the addressing scheme, the resolution levels that are included in the set, and how these relate to zoom levels in the map client, and the projection of the map. A tiling scheme can be global, covering the whole surface of the world, or it can cover some smaller area such as a country. For example, Google Maps, Bing Maps and Yahoo! Maps all use a global logical tile scheme based on the Mercator projection. Defining a logical tile scheme for the Mercator projection is simple because the Mercator projection projects the earth into a square surface. The smallest scale surface consists of one tile, the geographical extent of which, is the whole world. This tile is then divided into four equally sized square tiles for each consecutive resolution level. This type of scheme is also referred to as a tile pyramid (*Figure 2*). For this tile scheme tiles are addressed beginning from the top left corner, by indicating the row and column number, starting from 0. The address of the top-leftmost tile is thus 0, 0. The tile below this first tile has an address of 0,1, and the tile on the right side of the first tile has the address 1,0. By adding the resolution level in front of the address the exact position and resolution level of the tile can be identified by a map client. Another example is a scheme

based on a geodetic projection where the earth is portrayed as a rectangle that is 360 degrees wide and 180 degrees high. This rectangle can be divided into two perfect squares, which results in the first resolution level having two tiles. The following resolution levels, then divide these two rectangles into four new squares each, in the exact same manner as in the case of the Mercator projection. (Sample & Ioup, 2010)

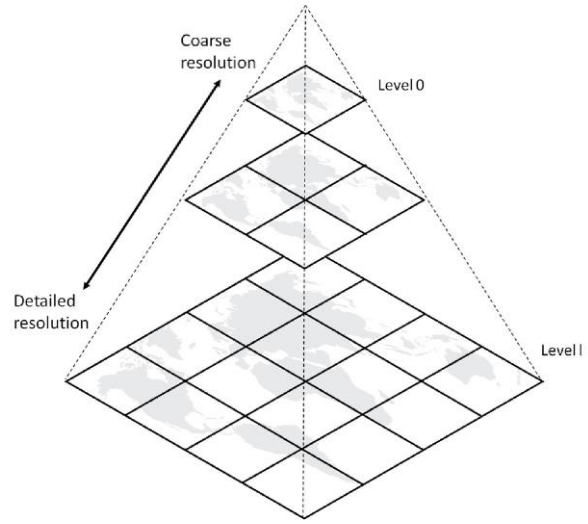

Figure 2. The tile pyramid (García, et al., 2012).

A logical tile scheme simplifies calculations on how many tiles are required to be rendered for a tile matrix set (Sample & Ioup, 2010). This will not only provide the map maker with an estimate of how much space is required for tile storage, but it can also help to estimate the time that it takes to render the set, as rendering whole sets of tiles is often time consuming. For example, the tile pyramid would include four to the power of n tiles in each tile matrix, where n is the number of the resolution level, beginning from zero for the coarsest resolution level. The number of tiles required for each resolution level in a tile pyramid consisting of sixteen zoom levels is presented in *Table 1*. The table also presents the storage capacity required to store each tile matrix, where the average file size of a tile is 50 kilobytes. It is apparent that the size of a tile matrix set with multiple resolution levels may quickly lead to storage issues.

Table 1. Number of tiles at each resolution level in a tile pyramid.

Resolution level	Number of tiles	Storage capacity that is required (file size 50kb)
0	1	50 kb
1	4	200 kb
2	16	800 kb
3	64	3.2 MB
4	256	12.8 MB
5	1 024	51.2 MB
6	4 096	204.8 MB
7	16 384	819.2 MB
8	65 536	3.23 GB
9	262 144	13.11 GB
10	1 048 576	52.43 GB
11	4 194 304	209.72 GB
12	16 777 216	838.86 GB
13	67 108 864	3.36 TB
14	268 435 456	13.42 TB
15	1 073 741 824	53.69 TB

2.2.4 Tile Storage

Storing tiles is a central issue in tiled map making, especially when rendering large tile sets containing millions of tiles. There are two major questions to be answered. First, should tiles be rendered and stored in advance, or should they be rendered on-demand (Quinn & Gahegan, 2010)? Second, what type of tile storage systems should be used (Sample & Ioup, 2010)?

Tiles can be rendered either in advance, or on-demand (Sample & Ioup, 2010; Quinn & Gahegan, 2010). It is also possible to render only a small selection of tiles in advance, while remaining tiles are rendered on-demand (Quinn & Gahegan, 2010).

Rendering tiles *in advance* will typically result in map clients performing well with regards to displaying tiles on the screen, as tiles can be fetched straight from the storage (OGC, 2010). Tiled mapping systems that create tiles by cutting larger images into fragments typically render all tiles in advance (Sample & Ioup, 2010). There are two reasons for this. First, reformatting, scaling and re-projecting images is often time con-

suming. Second, lower resolution tile matrices are often created from higher resolution matrices, which means that the higher resolution matrices have to be completely created before lower resolution levels can be viewed. The disadvantage of rendering in advance is that it may take a long time to render large tile sets, and the approach requires a lot of storage space (Sample & Ioup, 2010; Quinn & Gahegan, 2010). The method is not suited for maps that require constant updating, such as daily temperature maps or rainfall maps (Quinn & Gahegan, 2010).

The opposite approach is to render all tiles *on-demand* as the map client requests more tiles. This approach saves both time and storage space compared to rendering tiles in advance. The disadvantage of this approach is that it might be slow and frustrating for the user to wait for the tiles to be rendered. Even if tiles are stored after the first user has viewed them, the initial user still has to wait for tiles to be rendered. (Quinn & Gahegan, 2010)

Another option is to *compromise* and render some tiles in advance, while the rest are rendered on-demand. The key in this approach is to try and predict the most popular areas and render those upfront, while leaving less popular areas to be rendered on-demand. The method consists of two stages. First user behavior is monitored to determine the most popular areas. Second, the most popular tiles are rendered in advance. The disadvantage of this approach is that it assumes that user behavior does not change with time. It also requires the initial users to endure poor performance as they are monitored for behavioral data. Quinn and Gahegan (2010) compared rendering times of tiles from a tile matrix set that were predicted to be most frequently used with rendering times of the whole tile matrix set. The results of this comparison show that while their model required 62.7% less tiles to be rendered, the total rendering time was reduced by 41.5%, and 52.5% of total storage space was saved. (Quinn & Gahegan, 2010)

Tiles are stored in *tile storage systems*. Even when tiles are rendered on-demand, they are typically stored afterwards. This way, the user who navigates the map to the same location the next time, will have tiles served faster. When it comes to tile storage, there are several possible approaches. Tiles can be rendered into separate files, with a single-file-per-tile approach, into tables by using a database management system (DBMS), or into a single file. (Sample & Ioup, 2010)

Storing each tile in a separate file is perhaps the simplest approach. A folder structure based on, for example, resolution levels and columns can be used to store and organize files into multiple folders. This type of *single-tile-per-file approach* has several advantages. First, tiles are simple to address, for example by naming each tile with its row number, and storing that file into a folder named after its column number, which is itself stored in a folder with the zoom level of the tile matrix. Second, it is easy to update tiles in such a system without affecting the rest of the system. Third, this type of folder structure makes it simple to host tiles on a web server.

The disadvantages of the approach are related to its ineffectiveness. First, files use storage space ineffectively because they are stored in fixed sized blocks, but may contain any number of bytes. For example, a common block size is 4 096 bytes and storing a 50 000 byte tile in such a file would take up 12 blocks which are combined 53 248 bytes. Second, creating copies of the tile set can be time consuming, because it requires a separate file access and file write for each tile. Third, with regards to writing and reading tiles, the method does not perform as fast as storing multiple tiles in a single file. In addition Sample and Ioup (2010, p. 120) recommend such an approach to be used "when inherent properties of the file system are actually needed", for example, if tiles are updated frequently. (Sample & Ioup, 2010)

Another approach is to store files in a table in a *DBMS*. DBMSs are discussed in more detail in section 2.3.3 of this chapter. A row in the table would represent a single tile, while columns would represent information such as the image data and addressing. The advantage of the DBMS approach is that tile search time could be reduced by creating indexes for the address columns and storage can be more efficient than for a single tile per file approach. However, DBMSs can be expensive and time consuming to setup, configure and manage. In addition, they have many features that are not likely to be needed. Sample and Ioup (2010) recommend the use of a DBMS for storing tiles in two cases: first, when forced to use a DBMS; for example, web hosting services do not offer rights to a file system, only read/write access to a database. Second, when advanced query functionality is required, for example, if the tile metadata includes additional information, for example names and dates, and this information needs to be queried. (Sample & Ioup, 2010)

The third approach is to create a custom file format that is able to *store multiple tile images within the one file*. A custom tile index is required to locate tiles in the files. While the approach can outperform the single-tile-per-file and DBMS approaches, updating tiles in such a file can be complicated and result in fragmented files with old tiles. In addition, HTTP servers will require custom modules in order to read such a file. (Sample & Ioup, 2010)

Sample and Ioup (2010) compared the performance speed of the three tile storage system approaches when writing and reading tiles. Their results indicate that storing tiles into a single file outperforms the single-tile-per-file and DBMS approaches, both when writing and reading files. The single-file-per-tile approach is faster than the DBMS approach when writing data, but slower, when reading random tiles. However, when reading tiles and using tile-caching, the single-file-per-tile approach outperforms the DBMS.

2.2.5 Tiled Mapping Systems

Sample and Ioup (2010) define tiled mapping systems to have four core properties. First, map views have multiple discrete zoom levels that each correspond to a fixed resolution level. Second, a map view is built up from multiple image tiles. Third, the tile client accesses the images through a discrete addressing scheme. Fourth, tiled images are processed as far as possible in advance to minimize the processing work required as tiles are sent from the server to the map client. In addition Sample and Ioup (2010) define tiled mapping systems to have three optional properties. First, tile addressing may follow a single global projection. Second, the tile based mapping system may use a server and client based tile distribution system. Third, tiles may be organized into a few layers.

According to Sample and Ioup (2010), the process of rendering new tiles, using vector and raster source data, has five steps. First, a tile is chosen to be rendered. Second, the bounding box of the tile is determined. Third, data for the tile is queried. Fourth, the data is rendered to a bitmap of the proper size. Fifth, the bitmap is stored, with the file name indicating the tile address, or in case the tile is stored within a DBMS or a file with several tiles, the address is stored as an attribute of the bitmap.

In practice, larger tiles called metatiles are often rendered first. The metatiles are then cut into smaller tiles. This can significantly improve performance of the rendering.

When rendering new tiles from vector source data, objects such as text are sometimes cut at the tile borders because the point representing the location of the text is only located in a single tile. When rendering tiles, the rendered tile does not have information about objects in the neighboring tile, so the parts of the text that should be rendered on tiles, other than the tile containing the point location of the text, are not rendered. To overcome this problem, tiles are often rendered using a buffer around the tile, to ensure that all objects that are visible in the tile appear correctly on the tile. When a buffer is used, often a metatile is rendered first. A metatile ensures that the total buffer area of the image tile set that is rendered is significantly reduced, compared to rendering all the tiles individually.

2.2.6 Standards for Tiled Maps and Mapping Systems

OGC has released a Web Map Tile Service implementation standard (WMTS) that defines several properties for tiled systems and tile matrix sets. The WMTS builds on the preceding WMS Tile Caching implementation standard (WMS-C). WMTS-based systems are required to have a coordinate system and pre-defined resolution levels, unlike systems based on the Web Map Service implementation standard (WMS). Several well-known scale sets are defined in the WMTS. Map clients might not be able to rescale images or re-project them. A well-known scale set is a combination of both the resolution levels and the coordinate reference system (CRS) that is used. Each tile matrix set has a well-known scale set. A common well-known scale set will make two tile sets compatible from a technical perspective, although the styles of the maps may not be compatible. (OGC, 2010)

The Finnish JHS 180 recommendation, which builds on the WMTS standard, includes recommendations on tiled maps, similar to those of well-known scale sets in the WMTS standard. However, the JHS 180 concerns only maps that cover the area of Finland. The coordinate reference system to be used is ETRS89 / TM35-FIN. Resolution levels are given in *Table 2*. JHS 180 also recommends the size of tiles to be 256 x 256 pixels. The scheme is not global and thus has a top left corner position of 548 576 meters east and 8 388 608 meters north, in ETRS89 / TM35-FIN. (JUHTA, 2013)

Table 2. Resolution levels, pixel size, and height and width of tiles in JHS 180. (JUHTA, 2013)

Resolution levels	Pixel size (m)	Height and width of tiles (m)
0	8 192	2 097 152
1	4 092	1 048 576
2	2 048	524 288
3	1 024	262 144
4	512	131 072
5	256	65 536
6	128	32 768
7	64	16 384
8	32	8 192
9	16	4 092
10	8	2 048
11	4	1 024
12	2	512
13	1	256
14	0.5	128
15	0.25	64

The tile scheme from the JHS 180 recommendation consist of sixteen resolution levels (*Table 2*). At the highest resolution level, each pixel covers an area of 0.25 x 0.25 meters. Each tile, sized 256 x 256 pixels, covers an area of 64 x 64 meters. *Figure 33* depicts the grid that is formed by the JHS 180 recommendation for each resolution level. It is evident from viewing the grid that the extent of the area covered by the recommendation is much larger than the area of Finland.

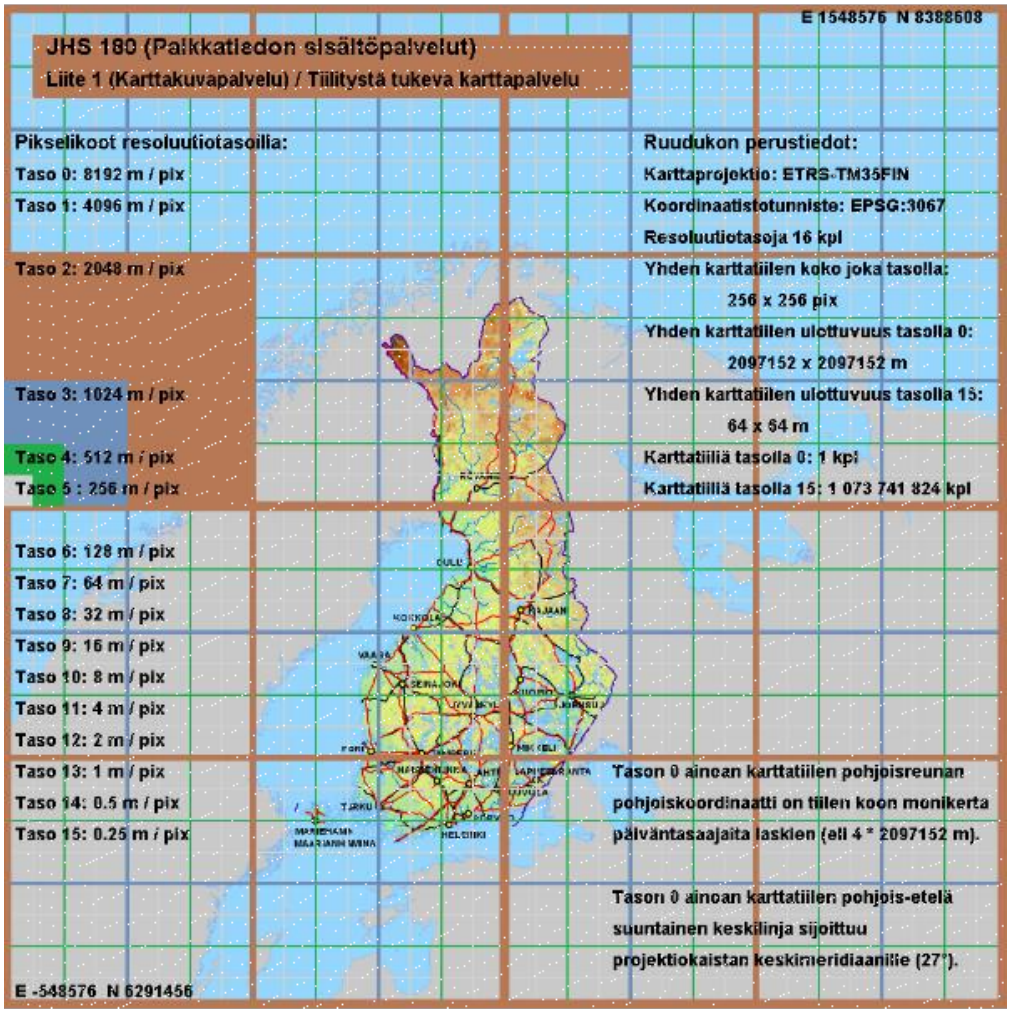

Figure 3. The JHS 180 grid. (JUHTA, 2013)

2.2.7 Vector tiles

In this thesis tiled raster maps are focused on. However, techniques for tiled vector maps have been increasingly developed in recent years. Gaffuri (2012) states that the next challenge in web-mapping is to improve vector web-mapping. Vector web-mapping would allow users to directly interact with map objects (Corcoran, et al., 2011; Gaffuri, 2012) and enable spatial analysis and adaptive visualizations on the client side (Corcoran, et al., 2011). Vector tiling is relatively new in web-mapping compared to raster tiling (Gaffuri, 2012). However, several tiled vector map solutions already exist (Dufilie & Grinstein, 2014). Vector tiles can adapt many techniques from raster tiling, for example vector tiles can use the same tile schemes, including grids and addressing schemes (Gaffuri, 2012).

2.3 Software for Tiled Map Making

Software, both FOSS and proprietary, for tiled map making is today widely available. Software for tiled map making comes from multiple software categories and may include a large variety of functionality, including functionality that is not needed in map making. On the other hand, some software products only have some of the required functionality for making maps, relying on other software products for the remaining components.

2.3.1 The Role of Software in Tiled Map Making

Software has a key role in each phase of tiled map making processes. When making decisions on software for different tasks in the processes it is critical to carefully assess the available options. Software impacts, for example rendering speed, cartographic capabilities and tiling scheme options. The developer has to make a decision between developing software components from scratch or using existing commercial and FOSS components (Longley, et al., 2015). Developing software from scratch will enable the developer to have full control over the functionality of the components (Longley, et al., 2015). However, the development process can be costly, demand considerable investment in time and possibly high level programming skills to produce a desirable outcome (Steiniger & Hunter, 2013).

As FOSS software standards have become more widely available and adopted, building software out of existing components has become an increasingly desirable alternative (Longley, et al., 2015). Today there are hundreds of GIS computer programs to choose from, with over 100 commercial software claiming to have mapping and GIS capabilities (Longley, et al., 2015) and more than 350 FOSS projects listed at freegis.org, a website that tracks FOSS projects (freegis.org 2015).

Searching through the functionality of such a large selection of software may be daunting and time consuming without comprehensive familiarity of the GIS software field. To help developers in their search of finding suitable software for systems, categorization of GIS software can narrow down the options. For example, Longley et al. (2015) categorize commercial GIS software into seven groups, desktop, web mapping, server, virtual globe, developer, mobile and other type of GIS software. Steiniger and Hunter (2013) on the other hand categorize FOSS into nine groups, remote sensing software,

desktop GIS, mobile GIS, exploratory spatial data analysis tools, spatial DBMSs, web map development frameworks, web map servers, server GIS and WPS servers, libraries and development tools. Steiniger and Hunter (2013) have created a map of major FOSS4G projects in 2013 (*Figure 4*). From the map it can be seen that placing computer programs within the categories is not simple, and that software may be part of more than one category.

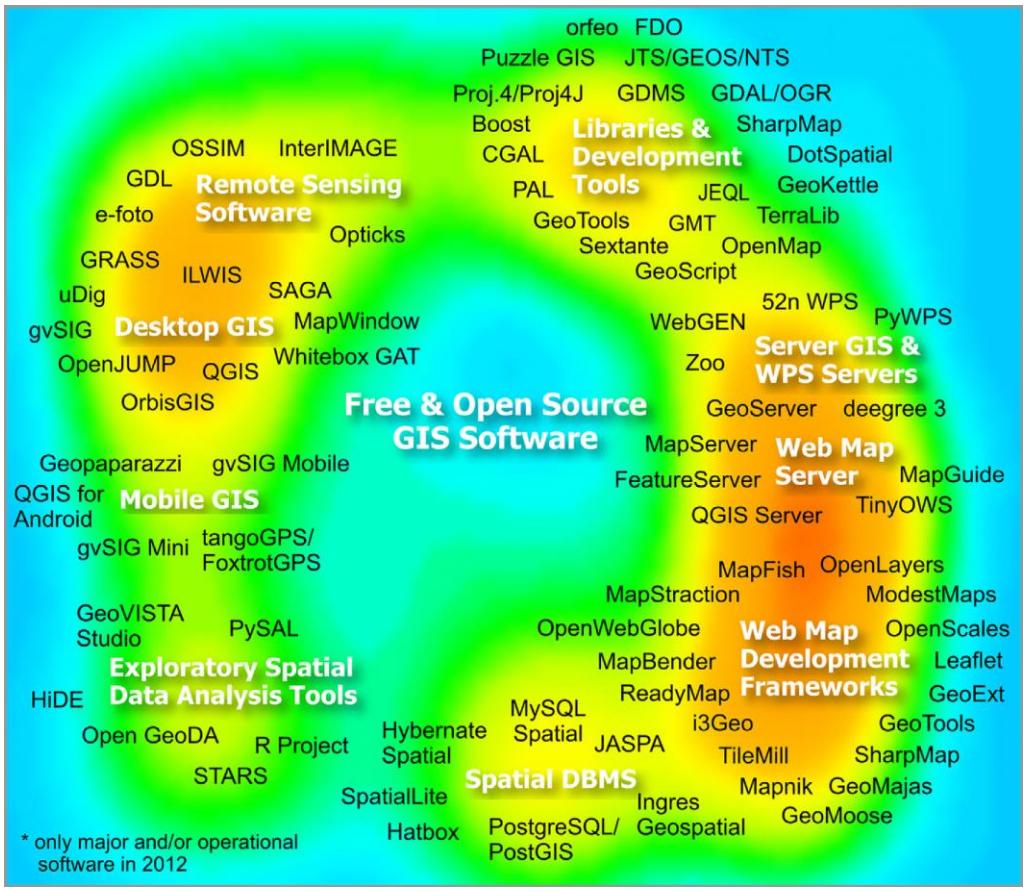

Figure 4. FOSS software categorization by Steiniger and Hunter (2013).

2.3.2 Free and Open Source GIS

FOSS has only recently become widely used in the field of GIS (Roth, et al., 2014), despite having been available for decades. Development of the earliest open source GIS, the desktop GIS program GRASS, began in 1982 (Neteler, et al., 2012), and the project is still active. FOSS software with GIS functionality is also referred to as FOSS4G, as in free and open source software for GIS (Steiniger & Hunter, 2013). Projects such as QGIS, PostGIS and OpenLayers have been particularly influential in the shift towards FOSS4G, as they have been able to attract large user bases (Steiniger & Hunter, 2013).

In FOSS, “free” is sometimes linked to free-of-cost, although the meaning of the term “free” in this context is freedom, as in the freedom to run the software for any purpose, freedom to study and adapt the software for own needs, the freedom to redistribute the software and the freedom to improve the software and to release the improvements to the public (Steiniger & Hunter, 2013). FOSS software is typically released under a Free Software license that ensures users the right to these freedoms (Neteler, et al., 2012).

2.3.3 Desktop GIS

Desktop GIS is mapping software that is installed and run on a single personal computer, and are not accessed or controlled from a different computer through a server (Steiniger & Hunter, 2013). They are widely used, in a large variety of industries (Longley, et al., 2015), and are probably the most commonly used GIS software (Steiniger & Hunter, 2013). Desktop GIS can vary from simple viewers to powerful mapping and GIS software with a rich variety of functions (Longley, et al., 2015). Functionality of desktop GIS includes displaying, querying, updating, editing and analyzing geographic information (Steiniger & Hunter, 2013). It is common for vendors to offer a few versions of their desktop GIS with different functionality levels and pricing. A basic version of the product might include functionality for viewing and exploring data, another version might add editing functionality, while the most advanced versions could include functionality such as data analysis and map creation (Steiniger & Hunter, 2013). Some advanced desktop GIS offer all required functionality to build complete tiled mapping processes.

Many desktop GIS products are built around a GUI which can slow down operating considerably if used to display large amounts of data. However, it is typically possible to operate desktop GIS without using the GUI for heavy tasks. According to Steiniger and Hunter (2013) there are at least eight FOSS desktop GIS products that have functionality comparable to commercial GIS software, have a commercial service provider that offer software support and have an internationally focused user and developer community. FOSS desktop GIS products typically focus on some particular range of functionality which results in many FOSS desktop GIS users to use several different systems for different tasks (Steiniger & Hunter, 2013).

2.3.4 Spatial Database Management Systems

Database management systems (DBMSs) are often effective tools to store and manage large databases. There are three types of DBMSs: relational DBMSs (RDBMS), object DBMSs (ODBMS), and object-relational (ORDBMS) DBMSs. RDBMSs use two-dimensional tables to store data and their attributes. The main weakness of an RDBMS is that it cannot store object state and behavior. For this, an ODBMS is needed. The success of ODBMSs has been suppressed by RDBMSs already been widely in use when ORDBMS first appeared. Vendors added important elements of ODBMSs to their RDBMSs, forming a hybrid version of the two, an ORDBMS. (Longley, et al., 2015)

Spatial extensions make it possible to store geographic information into regular DBMSs. DBMSs with spatial extensions are also referred to as spatial DBMSs. Spatial DBMSs typically provide spatial indexing structures. Compared to other categories of software, there are relatively few FOSS projects that develop spatial extensions for DBMSs. Despite their small number, FOSS spatial DBMSs have become popular and have large user communities. (Steiniger & Hunter, 2013)

Tiled mapping systems can use DBMSs to store source data, process data and to store tiles. However, benchmarking of DBMS performance shows that using one is not always justified for all phases of the process. Benchmarking of tile generation and tile retrieval, into and from, a single file, multiple files and a DBMS, shows that the DBMS approach is the slowest among the three for tile storage, and comes in second, behind the single file approach, in tile retrieval (Sample & Ioup, 2010).

Nonetheless, DBMSs also have several strengths, compared to storing data in files on a hard drive. For example, the maintenance costs are typically lower because of better organization of data and reduced data duplication. Applications can be data-independent, meaning that they do not have to store the data themselves, rather they can use data directly from a database. User knowledge can be transformed more effectively. Security and standards for data and data access can be established and enforced. DBMSs are also able to manage large numbers of concurrent users working with vast amounts of data better than file based systems. (Longley, et al., 2015)

However, DBMSs also have other weaknesses to consider. DBMSs can be unnecessarily complex for small projects (Longley, et al., 2015; Sample & Ioup, 2010). As such, a

spatial DBMS is often used for systems that require high performance, that include large amounts of data (Steiniger & Hunter, 2013). Another weakness of a DBMS is the potentially high cost of acquiring, setting up, configuring and maintaining the system (Sample & Ioup, 2010)

Yet another weakness of DBMSs is that single user performance will often be better in a file based system, especially for more complex data formats and structures with indexes and access algorithms (Longley, et al., 2015).

2.3.5 GIS Developer Toolkits and Libraries

Developer toolkits and libraries provide developers with ready built software components to be used in their own systems (Steiniger & Hunter, 2013). The components are integrated in the developer's own system with bindings in different programming languages. Development toolkits and libraries can have very different functionality. For example, Proj.4 is a FOSS projection library that enables coordinate transformations (Proj.4, 2015). Proj.4 is written in C/C++, and has been ported to other programming languages such as Java and JavaScript. Libraries such as Geospatial Data Abstraction Library (GDAL), offers tools for reading and writing geographic data, and are today used as components in many software systems (GDAL, 2015a).

Other types of components that are widely used are plug-ins that extend the functionalities of other systems. Plug-ins are typically available for all major software applications ranging from desktop GIS to web map development frameworks. It is noteworthy that GIS libraries and toolkits are sometimes used in non-GIS systems, and non-GIS libraries and toolkits are often used in GIS systems.

For tiled map making processes, developer toolkits and libraries can often prove to be useful. For example, GDAL is able to make data conversions that might be required when processing data for maps, and the Proj.4 library or one of its ports can be used when building map client applications for maps in local coordinate systems. Also, because the components offer code bindings, it is usually easy to incorporate them into automated processes. The nature of these components and their wide usage can sometimes make it difficult for developers to be aware of that they are using some toolkit or library when they are using it through another application.

2.3.6 Web Map Development Frameworks

Web map development frameworks enable developers to quickly build map clients. Map clients display maps for users to read and interact with. Since Google Maps was released in 2005, clients that support tile based maps have increased rapidly. Map clients have to perform three tasks to display tiled maps (Sample & Ioup, 2010). First, they have to calculate which tiles are required to fill the map view. Second, they have to fetch the tiles. Third, they organize the tiles on the screen for the user to view.

The first task that a map client is required to perform in order to display tiles is to calculate the indexes of tiles that fill the map view. The task requires implementation of a function that receives the extent of the map view as input, and returns the set of tiles that are required to fill that view as an output. If the map client uses discrete resolution levels, the function is simple. In this case zoom levels on the map client can directly match the resolution levels of the tiles. As the user zooms in and out, the client loads the tiles from the next or previous resolution level. The client then only needs to know the origin of the map view, the index of the tile is located at that point, and how many tiles are required to fill the map view. At no point does the map client need to know the geographic extent of the map view.

However, when the client uses continuous resolution levels, the task is more complicated and the client will need to know the geographic extent of the map view. First the resolution level needs to be determined as either degrees per pixel (DPP) or in cartographic coordinate systems, as coordinate units per pixel, for example meters per pixel (MPP). After this the resolution level of the map view can be compared to the resolution of the tiles. The resolution equals its geographic height and width divided by its height and width in pixels. The resolution level of the map view will mostly have a value between two tile resolutions. According to Sample and Ioup (2010), the right resolution level to choose is the one for which the tile resolution is higher than the resolution level of the map view, unless the difference is less than 10% to the lower resolution tiles. A higher resolution tile is more detailed and looks sharper, but the performance increases when using lower resolution tiles. (Sample & Ioup, 2010)

Once the map client has worked out which tiles it needs, it can move on to its second task, retrieving those tiles. The retrieval of tiles is greatly influenced by the method with which the tiles are stored, for example, if tiles are stored locally in a DBMS, the map

client connects to the DBMS and retrieves the tiles with SQL queries (Sample & Ioup, 2010)

The last task that map clients need to perform is generating the map view. As in the first task of calculating which tiles the map client needs to fill the map view with, there are two cases to consider, continuous and discrete resolution level view. Again a discrete resolution level will provide a simple solution, where the client only needs to place the images according to their indexes. The continuous resolution level view is more difficult, because the overall image needs to be resized for the map view. (Sample & Ioup, 2010)

2.3.7 Selecting GIS Software

Software selection processes help developers to make decisions on what software to choose for their systems. These processes aim to identify available options for tasks in the system, and to carefully asses the positives and negatives of the options. Cost, licensing and ease of use are also important aspects to consider in the process (Steiniger & Hunter, 2013).

There has been several proposed software selection processes for GIS software. Steiniger and Hunter (2013) propose a five stage selection process for selecting FOSS4G software (*Figure 5*). First, the developer creates use cases for the system. Second, the developer establishes evaluation criteria based on these use cases. Third, the developer evaluates the software. Fourth, the developer weighs the criteria. Fifth, the developer makes the decision on what software to use based on the results from the previous stages. Steiniger and Hunter (2013) also list nine questions about the system to be developed that should help the developers to narrow down the selection of software for a system.

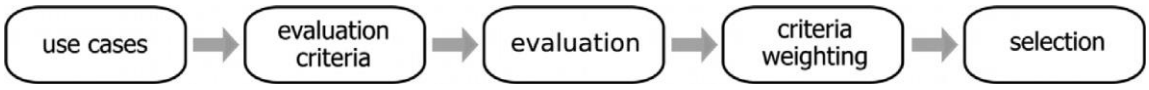

Figure 5. Steiniger and Hunter's (2013) software selection process.

Eldrandly and Naguib (2013) present a three phase software selection process for GIS software, consisting of a justification phase, a screening phase and an evaluation phase. The purpose of the justification phase is to justify the use of GIS software for the system. In the screening phase, software is searched to find suitable software for the sys-

tem. This phase has two parts, first, the required software capabilities are defined, and second, software with such capabilities are searched for. The third phase of the software selection process is the evaluation phase when software is compared to each other and additional criteria not used earlier in the screening phase is considered. The output of this phase is the GIS software listed in the suitability order.

According to Eldrandly and Naguib (2013) there are five essential evaluation criteria to consider when selecting GIS software for a system: cost, functionality, usability, reliability and vendor. Steiniger and Hunter (2013) state that, especially when deciding on the use of FOSS software, it is important to assess the size of the user community, and support from companies. These factors correlate with the availability of support and professional aid, including paid support. Such projects are also more likely to continue for a longer period of time.

2.4 Summary

This chapter discussed the theoretical and technical background on which the development of a process for producing tiled raster map series is based. In Section 2.1, map making in general was discussed, including the historical background of map making, map production processes, map series, and map design. In Section 2.2, tiled mapping was discussed, including logical tiling schemes, defining tile matrix set properties such as tile addressing scheme and projection; tiled images, the image fragments that tiled maps consist of; tile storage, including when to store tiles, and alternatives for tile storage systems; standards for tile based mapping systems, such as the OGC WMTS standard; and finally vector tiles. In Section 2.3, GIS software for tiled mapping, including software types such as, Desktop GIS, DBMSs, GIS developer tool-kits and libraries and web-map development frameworks, were discussed. Processes for selecting software for GIS systems were discussed at the end Section 2.3. The next chapter presents the materials and methods for the development of the map generation process, including a plan for how the development is divided into stages, the description of source data, and methods for selecting software that were used in the process.

3 Methods and Materials for the Development Process

In this chapter, methods and materials used during the development of the map generation process is presented. In Section 3.1, the planning of the development process is presented, including specifying and reviewing of the capabilities of the process that were desired by the FGI, sketching of a process model, and composing a plan for the remaining stages of the development process. In Section 3.2, source data used for the process is presented. In Section 3.3, software selection methods for the development process are discussed.

3.1 Development Process Plan

The aim of the development process was to build a map generation process for producing multi-scale tiled maps for with varying use contexts, by using FOSS tools. The outcome of the development process was then to be qualitatively evaluated, to discover the strengths and weaknesses of the process and to plan future development work for improving it. Planning the development process began by specifying and reviewing the requirements for the map generation process. To gain a better understanding of what was required from the development process, a simple process model was sketched of the map generation process. A plan was then written for how to divide the development of the process into stages.

3.1.1 Requirements for the Map Generation Process

The FGI desired an automated map generation process that could produce multi-scale, tiled raster maps with national extent. Initially, the process is required to produce three maps with different use context. The styles of the maps were to be based on the styles of the ‘Topographic map’, ‘Forest map’ and ‘Superficial Deposits map’ map types from the Viherkehä maps. The new maps were to have the same level of cartographic quality as the Viherkehä maps. The use of existing FOSS tools was encouraged. The use of proprietary software was to be considered only in a situation where no FOSS was found for a task. The process was required to use a tile scheme based on the Finnish JHS 180 standard with two exceptions: first, tiles with no data were not required to be rendered; second, resolution levels 0, 1, 14 and 15 were not required to be rendered.

The maps were to use topographic data derived from the NLS Topographic Database. Hillshading and contour lines for the maps were to be derived from the 2 m and 10 m

digital elevation models (DEMs) of the NLS. Maps were to use place names derived from the NLS 'Geographic names'. The 'Forest map' map type was to use the Corine Land Cover 2012 raster dataset, from which forest areas were to be extracted, and random points that represent the locations for tree symbols in the forest map, were to be generated within these areas,. The Corine Land Cover 2012 raster was also to be used in the 'Forest map' map type, for rendering forest covered areas with a green hue. The 'Superficial Deposits map' map type was to use the Geological Survey of Finalnd's (GTK) superficial deposits datasets of varying resolution.

3.1.2 A Map Generation Process Model Sketch

The development of the map generation process began by sketching a simple model of the process (*Figure 6*). The model was based on the requirements for the process and resembles the tiled map generation process used by the NLS (see Sipilä 2015). The process depicts how the map generation process transforms source data into tiled maps using two sub-processes. First, in a data processing sub-process the source data is processed and stored in new databases in a form that can be used directly to render tiles. Second, in a tile rendering sub-process, the processed data is used to render tile matrix sets.

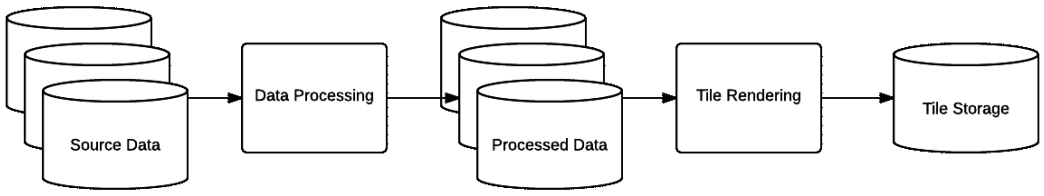

Figure 6. An early sketch of the map generation process.

The central element of the tile rendering sub-process is a tile rendering system that renders tiles according to user-defined map design and stores rendered tiles into a tile storage system. The tile rendering system is shared by all maps that are produced by the map generation process. However, maps with different use context use different source data and have different map design. The map generation process is extended with production flow lines for the new maps by adding data processing operations that become part of the data processing sub-process, and map design that is used by the tile rendering system.

3.1.3 Stages of the Development Process

Based on the sketched process model, a three stage plan for finishing the development process was composed (*Figure 7*). In the first stage of the process, the tile rendering system was to be built. In the second stage of the process, production flow lines were to be added to the map generation process. In the third stage of the development process, the finished map generation process was to be qualitatively evaluated, aiming to provide insight on what future development work should focus on. In addition, the qualitative evaluation was to evaluate how well the finished map generation process would satisfy the requirements set for it, and how effective the system is at rendering tiles.

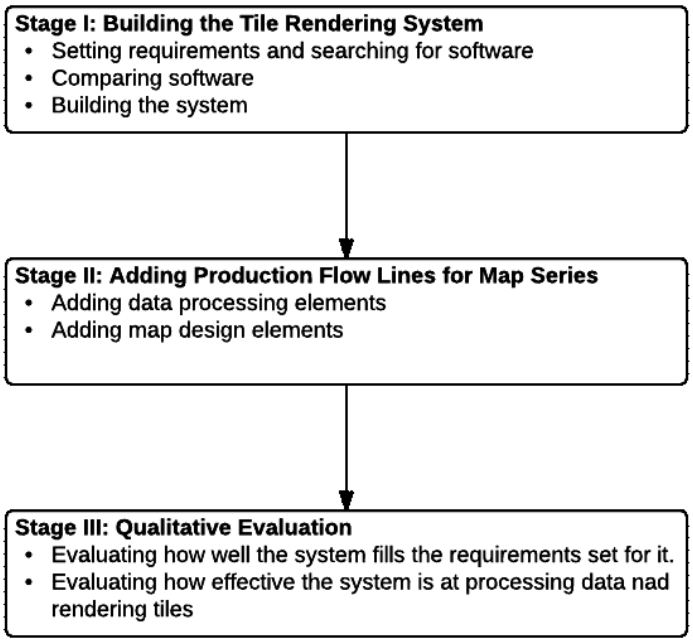

Figure 7. The three stage plan for the development process.

3.2 Source Data

The source data that is used in the map generation process was selected because it is the same data that was used in the Viherkehä maps, on which the styles of the new maps were also based. However, in the case of the Elevation model 2 m, and the GTK Superficial deposits datasets in the scale of 20 000, the data was not available from the whole area of Finland. Instead, the most detailed data that was available was used. This meant the addition of the Elevation model 10 m and coarser scale GTK Superficial deposits datasets.

3.2.1 The NLS PostGIS Databases

The NLS Topographic Database contains topographic features stored as vector data. The data covers the whole area of Finland. Its resolution has been expressed as having a scale of 1:5000 – 1:10 000. Roads and name features are updated continuously, building and border features are updated once a year and other types of features are updated approximately once every 5-10 years. The NLS Topographic Database is managed in a SmallWorld GIS spatial DBMS. Coarser scale datasets have been derived from the NLS Topographic Database. The data resolution in these datasets is expressed as having a scale of 1:100 000, 1:250 000, 1:1 000 000 and 1:4 500 000. The NLS uses the datasets for their digital and printed topographic map series. (NLS, 2015a)

The NLS produces topographic tiled maps that use the NLS Topographic Database, and the coarser scale topographic datasets as source data. In addition, the Geographic names of the NLS are used. The data is processed and stored into PostGIS databases before it is used for rendering (Sipilä, 2015). The databases were made available for the map generation process presented in this thesis. They include, a database for the topographic data, databases for each coarser scale data set, and a database that contains geographic names. Topographic data, derived from the NLS Topographic database, is also available in Geographic Markup Language (GML) and shapefiles. However, the shapefiles files do not follow the same data scheme as the PostGIS database and GML files and the data for both the GML files and shapefiles is divided into the files according to the TM35 map sheet division (see NLS, 2015b), which means that objects are cut at the borders of the sheets. The use of the PostGIS databases resulted in less need to merge objects that have been cut in order to divide them into files.

3.2.2 The NLS Elevation Model 2 m and 10 m

The NLS has two DEM products, the Elevation model 2 m and Elevation model 10 m. Elevation model 10 m is a DEM with a 10 x 10 meter grid size with elevation values in the N2000 elevation system. Elevation model 10 m has national coverage, and is the most accurate DEM covering the whole area of Finland. Its accuracy is on average 1.4 m. Elevation model 10 m is open data, and available to anyone in ASCII gridded XYZ format, divided into files according to the TM35 map sheet division. (NLS, 2015c)

Elevation model 2 m is a DEM with a 2 x 2 meter grid size with elevation values in the N2000 elevation system. Elevation model 2 m has been produced from lidar data. Datasets have two quality classes I and II. Quality class I data (coverage with green in *Figure 8*) has a vertical accuracy of 0.3 meters. Quality class II data (coverage with red in *Figure 8*) has an accuracy of 0.3 – 1 meters. Elevation model 2 m is currently in first-round production and is not yet available for the whole area of Finland, although new areas are continuously released. Quality class II data is also being replaced with quality class I data. Elevation model 2 m is open data and is available as ASCII format, divided into files by the TM35 map sheet division. (NLS, 2015d)

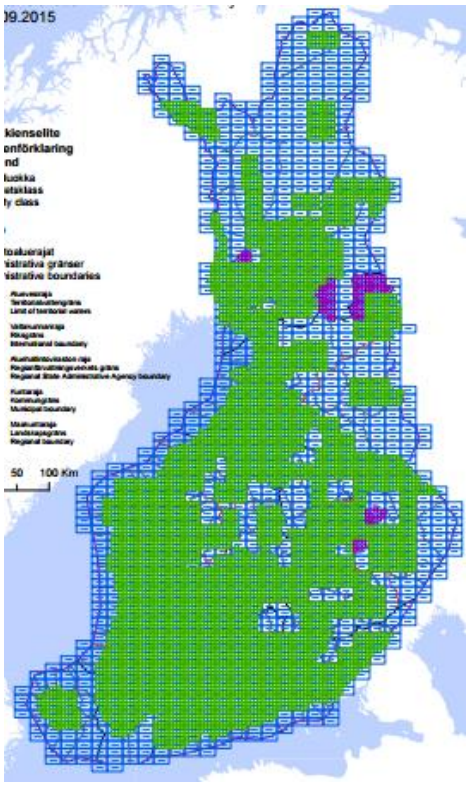

Figure 8. Coverage of the NLS Elevation model 2 m (30.09.2015) (NLS, 2015e).

3.2.3 Corine Land Cover 2012 Raster

The Corine Land Cover 2012 raster depicts land use and land cover of Finland in 2012. The raster has a pixel size of 20 m x 20 m. The data is categorized into five main classes of land cover, divided into 14 level two classes and finally divided into 44 level three classes. The Corine Land Cover 2012 is open data, and is available from the Finnish Environment Institute (SYKE) in GeoTiff format. (SYKE, 2014)

3.2.4 GTK Superficial Deposits Datasets

GTK provides superficial deposits data from Finland in vector format, in scales of 1:20 000 / 1:50 000 (GTK, 2015a), 1:100 000 (GTK, 2015b), 1:200 000 (GTK, 2015c), and 1:1 000 000 (GTK, 2013). The datasets are open data, and are available in ESRI File Geodatabase (FileGDB) and MapInfo TAB formats, with the exception of the 1:1 000 000 dataset, which is only available in FileGDB format.

The 1:200 000 and 1:1 000 000 datasets cover the whole area of Finland (GTK, 2015c; GT 2013), while the 1:100 000 dataset supplements areas that are not covered by the 1:20 000 dataset in southern Finland (GTK, 2015b). The 1:20 000 / 1:50 000 is the most detailed superficial dataset from Finland (GTK, 2015a). However, its coverage is limited to parts of southern Finland.

3.3 Software Selection

A tiled map making process may need a wide variety of software from multiple software categories. DBMSs with spatial extensions are used for storing and managing geographic data. Spatial DBMSs often include tools for data processing. This type of tools are also found in many other types of computer programs. Existing map clients, and development tools for building them can be useful for reviewing both tiled maps that are under development and the finished maps. Some desktop GIS, for example, ArcGIS, have the functionality required for building complete tiled map making processes, although the processes may have limited capabilities compared to more customizable options.

Selecting the best suited software components was a central challenge in the development of the map generation process. Decisions had to be made on what existing software to choose, or whether software should be built from scratch. These decisions could have a great impact on the final process. For example, software components used for the tile rendering system define, among other things, what options there are for tile schemes, image format, symbology that can be used in the maps, and the method by which map design is added to the process. Because of time limits, detailed selection processes, such as presented by Steiniger and Hunter (2013) and Eldrandly and Naguib (2013) was not possible to be done for all software that was used. A more agile and flexible software selection method was used, as described below.

Software selections in the development process began with defining compulsory requirements, requirements that define the minimum functionality and properties that have to be included in software components. In the screening process, existing software packages that fill these requirements were searched for in the Internet and from research papers. Discovered software packages were then compared using additional evaluation criteria relating to the functionality that could be useful to improve the map generation process, or the maps that the process produces. Comparing software packages for all components was considered to be an ineffective solution, because it is time consuming to study all functionality and properties of all alternatives. Instead, the impact that the required component has on the map generation process was assessed. Software packages were compared only when components were assessed to have a significant impact on rendering speed or cartographic capabilities of the map generation process. Existing comparisons between applications were also used to help decide between software, for example, recent research papers that compared software.

In some situations, software selections were influenced by previous selections. The order by which software is selected is then important. Ideally, all possible software package combinations would be compared. However, the amount of alternative solutions would make the selection process time consuming. Selecting software for the tile rendering system was the most important decision of the whole process, because the tile rendering system has the greatest impact on rendering speed and cartographic capabilities. Hence, the software components of the tile rendering system were selected first. In practice, tile rendering systems can be built with various combinations of components. Desktop GIS such as ArcGIS offer all tools required to build a tiled map generation process. Other solutions consist of several components that operate together. For alternative solutions of tile rendering systems that consist of several components, the rendering engine was selected first, and was then combined with the additional components that are required to build a complete tile rendering system. Comparisons between alternatives for tile rendering systems were compared by using the combined functionality of all components that were used for each solution.

4 The Tile Rendering System

This chapter presents the development of the tile rendering system. Before the system could be built, suitable software had to be first searched for, and the discovered alternatives had to be compared. In Section 4.1, the searching of software for the tile rendering system is presented. In Section 4.2, the comparison of the discovered alternatives is presented. In Section 4.3, the building of the tile rendering system is presented.

4.1 Software Alternatives for the Tile Rendering System

The first stage of the development process was to build the tile rendering system. The tile rendering system renders tiles from processed source data according to given map design. First, compulsory requirements for the software were composed, based on the functionality and properties required by the tile rendering system. Second, suitable software was searched for in a screening process. Third, the different alternatives were compared and a selection was made based on the outcome.

4.1.1 Compulsory Requirements

To find suitable software for the tile rendering system, all required functionality and properties of the system were first identified and listed (*Table 3*). The requirements were based on desired capabilities for the map generation process, see section 3.1.1.

Table 3. Table of Compulsory Requirements for the Tile Rendering System.

Type:	**Requirement:**
Tile options	Finnish JHS 180 recommendation compliant tiles (with some exceptions)
Tile options	Tiles in PNG format
Tile options	Tile storage following a single-tile-per-file approach, stored on disk.
Cartographic technique	Ability to render points with PNG and SVG icons.
Cartographic technique	Ability to render line and polygon patterns, in addition to rendering lines, polygons and raster datasets.
Cartographic technique	Ability to rotate text and symbols
Functionality	A map viewer where tiles can be viewed after they have been rendered
Other	Free and open source software

The compulsory requirements were as follows. Tiles are to be compliant with the Finnish JHS 180 recommendation, although the whole extent (bounds; x-min, -548 576; y-min, 6 291 456; x-max, 1 548 576; y-max, 8 388 608), and all resolution levels (0 - 15) of the JHS 180 recommendation were not required to be rendered. Only tiles that contained data, and resolution levels 2 to 13 from the JHS 180 recommendation were to be rendered. The five highest resolution levels correspond to resolution levels that were used in the Viherkehä map series. The tile size of 256 x 256 pixels is used and the CRS is ETRS89 / TM35-FIN. Tiles were to be rendered as PNG images into a single-tile-per-file folder structure. A tile viewer that displays the tiles as maps was also required, enabling style and configurations to be evaluated.

In addition, some minimum requirements were defined, related to the cartographic techniques that the rendering system software should be able to produce. In addition to point, line and polygon symbolizers, the renderer was required to be able to render, polygon and line patterns, and be able to use raster source data. Points were required to be able to use custom PNG and Scalable Vector Graphics (SVG) icons. Text and symbol rotation was also a compulsory requirement. The use of FOSS was encouraged, and was thus set as a compulsory requirement. If the search did not provide any results, then proprietary software could be searched for.

4.1.2 The Screening Phase

The range of possible solutions for a tile rendering system is diverse. Many existing commercial and FOSS software packages enable rendering of tiled maps. The software packages have vast differences in functionality and capabilities, and according to GIS software categorizations, may be found from several different categories. For example, in Steiniger and Hunters (2013) map of FOSS4G, FOSS that have the functionality to render tiles are included in Web Map Development Frameworks, Web Map Server, and Server GIS and WPS Server categories. Also, some desktop GIS such as ArcGIS are known to have all required functionality to render tiles. For most of these programs, rendering tiles is not their main functionality. This means that programs that are able to render tiles, typically include redundant functionality that is not needed in the map generation process.

Three solutions were discovered that fill the compulsory requirements set for software for the tile rendering system. These are solutions based on GeoServer, MapServer and Mapnik.

GeoServer is a widely used Java-based FOSS GIS server for sharing geospatial data (GeoServer, 2015a). It has been built on OGC standards, including WCS, WMS and WFS. During the time of the software selection, the newest stable version of GeoServer was 2.6.2. GeoServer 2.6.2 is able to render both polygon and line patterns and uses the OGC standardized styled layer description (SLD) stylesheet language (GeoServer, 2015b). Version 2.6.2 does not enable the use of compositing operations. Compositing operations enable altering the way colors and textures of layers interact with each other, for example, instead of rendering a layer on top of the destination image, the layer could be rendered under it, covering only pixels with no existing value (MapBox, 2015). Geoserver includes a built-in OpenLayers viewer for viewing tiles.

MapServer is a widely used FOSS platform for publishing spatial data on the web (MapServer, 2015a). At the time when the development of the map generation process began, and the comparisons of software for the tile rendering system were conducted, the latest version of MapServer was 6.4. MapServer has supported SLD since the release of version 4.2 (MapServer, 2014). MapServer 6.4 is able to create PNG images (MapServer, 2015b), and is able to store tiles on disk (MapServer, 2015c). MapServer 6.4 included a built-in OpenLayers viewer for testing styles (MapServer, 2015d). MapServer 6.4 did not support the use of compositing operations.

Mapnik is a map image rendering library. It uses the AGG and Cairo rendering engines for rendering tasks. The native style descriptor language of Mapnik is MapnikXML. Mapnik does not have a GUI and cannot be directly controlled from the command line. Instead, Mapnik uses Python and NodeJS bindings that enable developers to write scripts that initiate rendering of map images. To render tiles, the scripts need to handle the tile scheme, by initiating the rendering of each tile separately, providing Mapnik with the bounds of the tile and the name of the file where the tiles are stored. Mapnik does not provide map viewing functionality. Lacking inbuilt functionality for both producing tiles according to custom tile schemes and for displaying tiles on the screen, Mapnik requires additional components for it to be a viable choice for the tile rendering system. Screening for such software yielded two viable alternatives for controlling the

tile rendering, TileCache and TileStache. The two software are similar and enable tiled maps to be rendered according to given configurations, that include tile size, projection, file format, metatiling (TileCache, 2015; TileStache, 2015). There are also plenty of map viewers that can be used to present tiles. Building one with a map interface application is relatively fast, and several FOSS alternatives were found. Mapnik was considered a viable alternative coupled with either TileCache or TileStache and a map viewer developed with a map application framework. The next step was to choose whether to use TileCache or TileStache, and then to choose which web map application framework to use to build a map viewer.

TileCache and TileStache were tested for the task. TileCache could be set up to render Mapnik images quickly, while TileStache proved to be much harder to implement, and no images could be rendered during the time that it was tested. It was concluded that TileStache was much harder to set up to render tiles with Mapnik. The software had the same functionality as far as was required by the tile rendering system. Both were able to produce tiles in custom schemes, including custom projections and custom tile sizes. Both software allowed metatiling and options on the metatile grid size. Both enable a buffer area to be created around tiles that is then cut from the final tile that is stored. Both enable storing tiles to disk as PNG files. Further analysis on the differences was considered to be unnecessary, and TileCache was selected because it was easier to implement. The next step was to select the map viewer.

A recent study by Roth et al. (2014) compared four well-known web map interface development frameworks, Leaflet, D3, Google Maps API and OpenLayers. The software packages were compared by creating a scenario with fixed requirements and a limited time period for participants to implement the scenario using the technologies. They then asked the participants about their emotional experiences of using the given technologies. The study showed that Google, the only closed source technology met 31 of the 48 requirements set in the scenario, while Leaflet came in second with 29 requirements, followed by D3 with 22 requirements and finally OpenLayers with 21 requirements. The study also shows that the users experienced most positive emotions when using Leaflet, followed by Google, then D3 and finally OpenLayers. The users felt more positive emotions than negative with Google and Leaflet while D3 and OpenLayers made the users feel more negative emotions.

Because the viewer has no impact on tiles produced by the map generation process, selecting the viewer was made based on a quick analysis of the three alternatives that had been discovered. Both the OpenLayers solution and a Leaflet solution were tested with test tiles and the tiles would load much faster on screen in Leaflet. D3 was considered too time consuming to implement such a test with it. No further comparisons were conducted and Leaflet was chosen as it was considered to be superior to the others due to faster loading of tiles and more favorable results from the study by Roth et al. (2014).

Although three acceptable solutions were found, many other noteworthy FOSS projects with tile rendering functionality were also discovered in the screening phase. One of these applications is TileMill. Tilemill was developed by MapBox for rendering tiled maps by using spatial data in shape files, PostGIS or SpatiaLite databases. TileMill has a GUI where styles can be created using TileMills stylesheet language, CartoCSS. TileMill uses Mapnik map rendering library to render map images. CartoCSS can be converted to MapnikXML, Mapnik's stylesheet language, in TileMill. However, MapnikXML cannot be converted back to CartoCSS. TileMill can only produce tiles in the Mercator projection. The production of TileMill has been discontinued by MapBox, which now maintains and develops a proprietary software called MapBox Studio that is based on the TileMill project. (MapBox, 2015)

4.2 Comparing Software Alternatives

4.2.1 Evaluation Criteria

The next step of the development process was to compare the three software solutions that had been found suitable for the tile rendering system. Because the selected alternatives included rich varieties of functionality, the software products were evaluated with a general method, by listing evaluation criteria categories and then identifying differences between the alternatives within each category. The evaluation criteria consisted of six categories: cartographic capabilities, rendering speed, customization options, ease of use, support, and maturity. The most important category of criteria in the evaluation was cartographic capabilities. The ‘Topographic map’, ‘Forest map’ and ‘Superficial deposits map’ map types used styles such as vignetting, and line objects that are cut by text of the same color. The most important cartographic capabilities were defined in the compulsory requirements.

Rendering speed was considered to be the second most important criteria in the evaluation. Tile rendering was expected to be time consuming, and long rendering times could make the whole process unusable. If rendering would take several weeks, the system would be too slow at producing maps effectively. In addition, slow rendering speeds make it slower to evaluate map design, and other elements of the process during the development. Additional categories of criteria, based on which the alternative solutions for the tile-rendering system were compared, included customization options, ease of use, maturity and support. None of these categories were considered as important as the ability to produce maps with high cartographic quality and fast rendering speeds. However, the additional criteria were to be assessed during the software selection process.

4.2.2 Comparison Results

The different software alternatives for building the tile rendering system were compared. The results are presented in *Table 4*. Based on the comparative analysis, a solution combining the use of Mapnik, TileCache and Leaflet was selected. The most critical factor to this was the comparison of cartographic capabilities. Mapnik was considered to be the best alternative in this category, because Mapnik was the only one of the three alternatives that enabled compositing operations, such as color blending. For example, using the compositing operation multiply when rendering hillshading on an image will multiply the values from the hillshading raster with existing values in the tile images. No other significant differences were found concerning cartographic capabilities.

The attempt to compare rendering speed revealed the difficulty of doing so. There are many factors that contribute to the total speed, including map design and source data that is used, and the method by which data is stored. The only method by which rendering speed could have been compared reliably would have been to build the complete system with all alternatives and to benchmark the speed of the finished systems. This type of testing would have been highly ineffective method to test the solutions and would have slowed down the development process significantly. Because of this, rendering speed was not compared.

Other evaluation criteria had little effect on the final outcome. All alternatives allow the tile scheme to be customized and tiles to be stored on disk. All components of all the three alternatives were considered to be mature enough to not suffer from problems re-

lated to early stages of software production. All alternatives also have good documentation and support, including wiki sites and forums where the problems and their solutions can be discussed. MapServer and GeoServer offer more complete solutions than Mapnik, requiring less additional components. However, TileCache has been designed to use Mapnik without additional bindings, only requiring Mapnik to be selected as the rendering engine in its configurations. This interoperability makes the combination of these two applications easy to use. The tile viewer would still have to be built with Leaflet from scratch.

Table 4. Evaluating Software for the Tile Rendering System.

Criteria:	**GeoServer**	**MapServer**	**Mapnik / TileCache/ Leaflet**
Cartographic capabilities			Compositing operations
Rendering Speed	Not compared	Not compared	Not compared
Customization options	Relatively few options	Relatively few options	Most customizable
Ease of Use	Relatively easy to use	Relatively easy to use	Most difficult to use
Support, including documentation, e-mails, forums, wiki etc.	Good support and well documented	Good support and well documented	Good support and well documented
Maturity	Mature	Mature	Mature

Based on the results of the comparison, Mapnik with TileCache and Leaflet was selected, because it offered compositing operations. The inclusion of compositing operations was considered to be of greater value than having slightly easier to use software.

4.3 Building the Tile Rendering System

The decision was made that the system would be built in a Linux environment using the Ubuntu 14.04 Desktop operating system. Ubuntu is a FOSS and the software components for the tile rendering system all run on Ubuntu. The tile rendering system was first built with Mapnik version 2.2.0, TileCache version 2.11, Bash scripting language, and Leaflet version 0.7.5. Mapnik was later updated to version 3.0.0. During the development of the tile rendering system, it became apparent that having the system render all areas of the maps at once was not a good solution for two reasons. First, the map generation process was to be developed further in the future to allow multiple hardware in-

stances to use it simultaneously to render different areas of the same map. Second, the method produced a large number of blank tiles before any tiles with data were generated, making it time consuming to test the process during its development. The system was configured to divide the processing of resolution levels 6-13 tile matrices into separate areas, according to the level 4 grid of the Finnish JHS 180 recommendation. However, only 40 of the 256 grid squares cover parts of Finland, so the system was set to render only these areas. These areas are referred to as rendering areas. Each individual area is referred to by names consisting of a letter and a number (*Figure 99*). From north to south, the rows in the girds that covers the area of Finland were named alphabetically beginning with the letter A. From west to east, where the squares cover the area of Finland, the letters were extended with a column number beginning from number one.

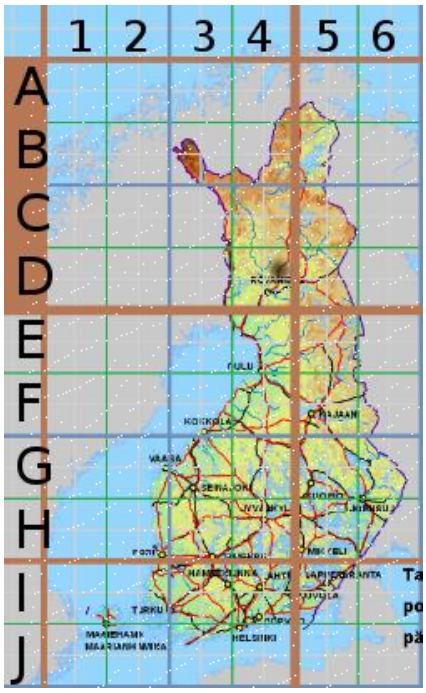

Figure 9. The given names of rendering areas in the JHS 180 level 4 grid.

Although setting the tile division up was not required to be automated, by scripting the process, it is easier to later edit the process for future needs. It also enables quick updating of all stylesheets within each rendering area, which is useful when testing map styles. A *Bash* script was written that first, reads a file containing the coordinates of each rendering area's bounding box. Second, the script creates a folder for each area. Third, the script copies the TileCache library into each folder. Fourth, the script writes a *tilecache.cfg* file that contains the configurations for TileCache, and a *GoogleDisk.py*

file that contains the addressing algorithm for TileCache for each area. The files are stored within the copy of TileCache. The GoogleDisk.py file that is included in TileCache version 2.11, where the tile addressing always begins from 0, is replaced by the new GoogleDisk.py file so that each tile is addressed as if the whole extent of the map was rendered at once. Fifth the script copies the MapnikXML stylesheets into the folders and rewrites the extent of the area in the stylesheets. Finally, the Bash script writes a new Bash script for each area that executes tile rendering for the area.

Resolution level 2-5 tile matrices are rendered simultaneously for the whole JHS 180 covered area. A similar Bash script is written that creates a folder for lower resolution levels, creates a copy of TileCache with the required configuration and addressing files, that copies the required MapnikXML stylesheets, and that finally creates the script that will begin the rendering. The time required for rendering these tile matrices is significantly shorter than the time it takes to render higher resolution tile matrices.

The tile viewer was built with Leaflet. While the tiles are being rendered, they are stored within a single-file-per-tile folder structure. Leaflet reads the PNG files from the folder structure, loads them, and displays the tiles on screen. The application includes tools for panning, zooming and selecting layers to be viewed.

5 Production Flow Lines for Maps

In this Chapter, the steps for building the map production flow lines for the three new maps is presented. In Section 5.1, the identifying of the data processing operations required for the maps is presented, in addition to the selecting of a spatial DBMS where the processed data is stored. In Section 5.2, the adding of the data processing operations to the map generation process is presented. In Section 5.3, the adding of map design to the map generation process is presented.

5.1 Required Data Processing and Selecting a DBMS

Adding production flow lines for map series was an incremental process where style definitions for maps were added to stylesheets one feature class at a time. In this process situations arose where the desired style or effect could not be defined with MapnikXML. However, many desired styles and effects could be achieved by first processing the source data. Reformatting of data was also required for source data that was only available in a format that are not supported by Mapnik.

5.1.1 Identifying Necessary Data Processing

Adding map design to the process revealed the need for several data processing operations. Required data processing that was identified is listed in *Table 5*. One option, which would have reduced the amount of data processing operations that were required, was to use Mapnik's capability of requesting vector data from a DBMS with custom SQL requests. However, complicated SQL requests that include data processing operations slow down rendering. It is not always necessary to process source data when tiles are rendered, for example, when the source data has not received any update but new tiles are rendered due to style changes. It was therefore concluded that it is better to build data processing operations for processing the source data, rather than using custom SQL requests when tiles are rendered. Because there may be situations where only some of the source data is updated, it was also concluded that it is sensible to build separate data processing operations for all types of source data.

The new maps were to use hillshading and contour lines that are derived from the NLS Elevation model 2 m and 10 m. Polygons depicting protected areas are also required to be processed because the desired style for them in the new maps was not possible to achieve in MapnikXML. In addition, most of the text features in the NLS PostGIS data-

bases are stored as point objects, each point representing a single letter. The font of the text and its size cannot be modified if the existing letter objects are used in rendering because the gaps between the letters are set for the font and text size used by the NLS in their topographic map series. To enable custom fonts and text size to be used the letters have to be merged together to form text objects representing complete names. The 'Forest map' map type was to use randomly generated points that mark locations of tree symbols within forest areas. The Corine Land Cover 2012 raster provided the required forest areas. Three different symbols are used for different types of trees: a spruce, pine and birch symbol. Each point that is generated is linked to information about which symbol it uses. Different forest types are given a different distribution of tree symbol types. For example, the mixed forest type is given 50% pine, 25% spruce and 25% birch symbols. Superficial deposits data was available only in FileGDB and MapInfo TAB formats, neither being directly compatible with Mapnik. The files are required to be reformatted into a format compatible with Mapnik.

Table 5. Required data processing.

Source Data:	**Target Data:**
NLS Elevation model 2 m and 10 m	Hillshading
NLS Elevation model 2m and 10 m	Contour lines
NLS Protected area polygons and borderlines	Desired type of protected area polygons
NLS Geographic names as letter objects	Geographic names as name objects
Corine Land Cover 2012	Points objects for tree symbols
GTK Superficial deposits datasets in FileGDB or TAB format	GTK Superficial deposits datasets in PostGIS, or other Mapnik supported format.

DBMSs are beneficial when datasets are large (Steiniger & Hunter, 2013). This applies to the data that is produced by some the data processing operations, and therefore the data is stored in a spatial DBMS if possible. However, Mapnik is not able to read raster files from a DBMS, and hence raster data has to be stored in files.

5.1.2 Alternatives for the Spatial Database Management System

Because a DBMS may impact the rendering speed of the tile rendering system, it was decided that the DBMSs should be selected carefully by first searching for software that would fill compulsory requirements, and then comparing all alternatives. Three compulsory requirements were set for the spatial DBMS (*Table 6*). First, the spatial DBMS was required to be compatible with Mapnik. Second, the spatial DBMS was required to be able to import all the data used by the tile rendering system. Third, the spatial DBMS was required to be FOSS.

Table 6. Compulsory requirements for spatial DBMSs.

Type:	**Requirement:**
Functionality	Compatibility with the Mapnik
Functionality	The capability to store all geographic data used in the process
Other	FOSS

The screening phase resulted in two widely used alternatives: PostgreSQL with PostGIS extension and SQLite with SpatiaLite extension. The most limiting factor to alternatives for the spatial DBMS was that Mapnik only supports reading from these two spatial DBMSs. Although, there are also relatively few FOSS spatial DBMS products compared to other software categories (Steiniger & Hunter, 2013).

PostgreSQL is a FOSS ORDMB that has been actively developed for more than 15 years. It runs on multiple operating systems, including Linux and Windows. The maximum database size is unlimited, enabling any amount of data to be stored in it. The maximum table size is 32 TB. (PostgreSQL, 2015). PostGIS is a spatial extension for PostgreSQL and includes a wide variety of vector and raster data processing operations (PostGIS, 2015).

SQLite is a DBMS aimed to be as lightweight as possible (SQLite, 2015). SpatiaLite extends SQLite with spatial DBMS capabilities and includes several data processing operations for both vector and raster data. (SpatiaLite, 2015)

5.1.3 Comparison of Database Management System Alternatives

The evaluation criteria used for comparing spatial DBMSs were: spatial indexing capabilities for tables and columns, availability of data processing tools, ease of use, support,

and maturity of the project. Two criteria were identified as being most important. First, spatial indexing enables faster fetching of objects from the DBMS and thus impacts the speed of rendering image tiles. Second, data processing tools of spatial DBMSs can be useful for the data processing stage, requiring less data imports and exports when the data can be processed within the system it is stored in. The other criteria were less important because they have less impact on the system.

The two DBMS alternatives were compared and the result of the comparison is presented in *Table 7*. PostgreSQL version 9.3, PostGIS version 2.1, SQLite version 3.8.8 and SpatiaLite version 4.2.1 were used in the comparison. The differences between the two spatial DBMSs are significant. It could be noted that the two DBMSs also aim at different goals. SQLite aims to be highly portable, and lightweight, with the cost of functionality, whereas PostgreSQL aims at having the highest level of functionality. PostGIS offers more options on spatial indexing than SpatiaLite. Also, PostGIS has far more data processing tools than SpatiaLite. There were no significant differences between the two spatial DBMSs in other evaluation criteria. Based on the comparison, PostgreSQL with PostGIS extension was considered to be the better alternative and was selected as the spatial DBMS for the map generation process.

Table 7. Comparing DBMSs

Criteria:	**PostGIS**	**SpatiaLite**
Spatial Indexing	More index alternatives	Less index alternatives
Data processing tools	More tools	Less tools
Ease of Use	No significant difference	No significant difference
Support, including documentation, e-mail, forums, wiki, etc.	No significant difference	No significant difference
Maturity	No significant difference	No significant difference

5.2 Data Processing

After the required data processing operations had been identified and a spatial DBMS had been selected for storing processed data, the next step was to build each data processing operation of the map generation process. Compulsory requirements for tools that were used for data processing were: included the required data processing functionality, included the possibility to be automated, were FOSS, processed the data in an acceptable amount of time.

5.2.1 Protected Area Polygons

In the PostGIS topographic databases received from the NLS, and the small scale databases, protected areas are stored as polygons and lines. The PostGIS topographic database also includes the borders of protected areas that are stored in a separate table as lines. In the new maps protected areas are symbolized as polygons that extend a fixed distance from the protected area borderline towards the middle of the area (*Figure 10*). Such a style was not possible to achieve in MapnikXML with the available source data. The NLS PostGIS databases include a database with similar polygons for protected areas. However, at some locations the polygons extend outside of the protected areas. *Figure 10* depicts this. On the left side in *Figure 10*, the NLS border polygon (dark green) can be seen overlapping (blue) the border lines of protected area (light green). On the right, the new border polygn that is generated stay within the protected area. The example is from the southern tip of Nuuksio National Park. In areas within the protected areas where the distance between two borderlines is shorter than the thickness of the buffer, the buffer will cross outside the protected areas.

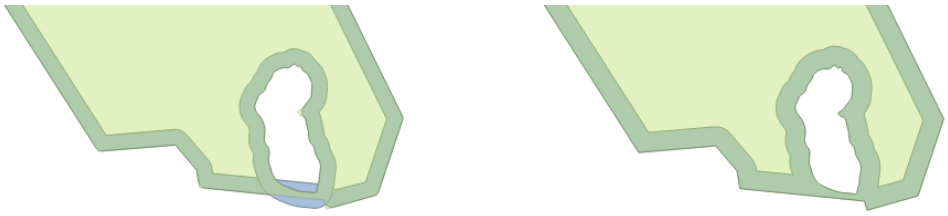

Figure 10. Existing border polygons (left), and the new ones (right).

The new protected area polygons are generated from the original protected area polygons and border lines in the NLS PostGIS databases. The necessary tools for the process were included in PostGIS. Importing and exporting data from and to a DBMS can take a significant amount of time. The decision was made to build the process with PostGIS without exploring other software alternatives. For protected areas in the small scale PostGIS databases, only polygons are required to produce the new protected area polygons. For these polygons a buffer is first generated around the edges of the polygon. The intersection of the buffer and the protected area polygon is then extracted to receive the new protected area polygon. The different resolution levels in the new maps have different requirements for the width of the protected area border, requiring the process to generate separate polygons for each level. For the protected areas in the topographic

PostGIS database, generating the new protected area polygons is more complicated because some of the existing protected area polygons are stored in fragments. The same operation would produce areas inside the polygon where the polygons are cut by the division into fragments. The protected areas stored as line feature have an attribute that tells whether the line is an actual protected area border, or if it is a line object that divides a protected area polygon into fragments. This enables the buffer to be produced only around the actual borders. However, it presents another problem: buffering the border lines creates several polygons for each protected area because the border lines are stored in segments. The intersection calculation between the protected area polygons and the buffers becomes slow because thousands of polygons are required to be compared to thousands of other polygons. Combining the buffers to create a single multipolygon is not enough to speed up the process. However, combining the protected area polygons and the buffers, each group of polygons as multipolygons and then calculating the intersection of the two multipolygons solves the speed issue. The resulting multipolygons can then be separated back to regular polygons. All the required SQL commands for the process were stored into a text file.

5.2.2 Text Objects

All text objects that are used as source data for the maps are stored in the NLS PostGIS databases. PostGIS offers all functionality required for merging letters to form names and other text features. The processing was tested, and PostGIS completed the required operations in an acceptable amount of time. No other tools were searched for this process.

All text objects have a unique identifier that is stored in the letter object attributes. The identifier enables letters belonging to the same text object to be collected into a single multipoint object. The multipoint is transformed into a single point and all the letter attributes of the letter point objects are merged in order from the point furthest to the west, to the point furthest to the east. The location of the point furthest to the west is selected to represent the location of the new text object. During the process, all relevant attributes of the letter objects are carried on to the text object. The SQL commands that are used in the process were stored into a text file.

5.2.3 Hillshading and Contour Lines

Hillshading and contour lines are generated from the NLS Elevation model 2 m and 10 m. Both DEMs were available as sets of DEM fragments, divided into files by the TM35 map sheet division. The 2 m DEM set was available in ASCII (ASC) file format. The 10 m DEM set was available in the ASCII gridded XYZ format.

The resulting contour lines are stored in a PostGIS database as line objects. Mapnik is not able to read raster data from PostGIS databases. Hence, the resulting hillshading raster layers are stored in GeoTiff format. Because the processing of the DEMs for large resolution levels is burdensome for the hardware and requires several interphases with long processing times, the DEM processing, for resolution level 6-13 tile matrices, was set to be divided into smaller areas. The same area division was used as for the rendering of tiles. The resulting hillshading raster layers and contour lines are stored separately for each area. Hillshading for resolution level 2-5 tile matrices are produced as part of the same process. However, a single raster is used for each resolution level that covers the whole area of Finland, and only the Elevation model 10 m as source data.

PostGIS was not considered to be an ideal alternative for DEM processing, requiring data to be imported and exported. Other solutions were searched for and the most suitable alternative that was discovered was GDAL. GDAL can be used for translating and processing geospatial raster and vector data. It has no GUI and is controlled from the command line. GDAL is used by over 90 other software for translation and processing tasks, including GRASS GIS, QGIS, Mapnik and PostGIS (GDAL, 2015b). The version that was used in the map generation process was 1.11.2. Other FOSS alternatives that were discovered included GRASS GIS and QGIS. However, QGIS uses GDAL for most of its raster processing, and GRASS GIS requires data to be imported before it can be processed, and the exported so it can be used for rendering with Mapnik. Therefore GDAL was selected as the raster processing tool. However, GDAL did not provide all the tools needed. One step of the process required more advanced image manipulation. For this step GNU Image Manipulation Program (GIMP) version 2.8.10 was used. No other FOSS software was found for that particular step. The different phases in the processing of the DEMs is depicted in *Figure 111*.

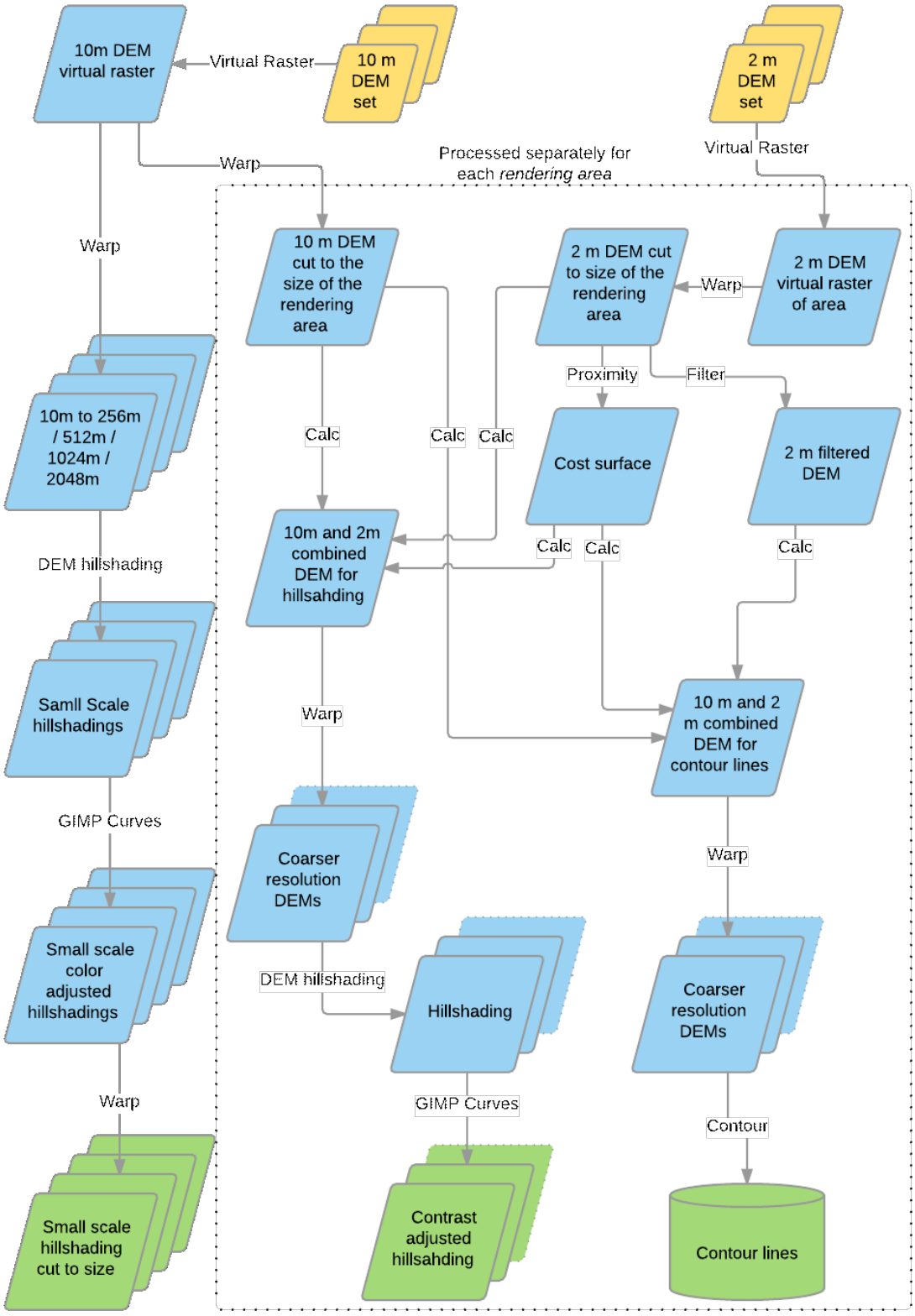

Figure 11. Processing of DEMs.

The processing of Elevation model 10 m begins with creating a GDAL virtual raster of the 10 m DEM set. For hillshading for resolution level 2-5 tile matrices, the virtual raster is resampled four times with gdal_warp producing four GeoTIFF files. The resampling is done with GDALs 'average' resampling method, and the pixel resolution of the resulting raster layers is: 256 meters / pixel, 512 meters / pixel 1024 meters / pixel and 2048 meters / pixel. The four coarser resolution DEMs are then used to create hillshading in GeoTIFF format with *gdaldem*. The hillshading layers that are produced are too dark to be used directly for the maps. The darkness is adjusted with the GIMP curves tools, for which a script was first written, containing the parameters of the curve. Processing the files with GIMP results in plain TIFF files. However, TIFF world files are created earlier during processing with *gdal_warp*. This enables most GIS software to recognize the files as GeoTIFF files. The final step to produce the small scale hillshading layers is to cut out all parts that overlap the area of Finland with *gdal_warp*. A polygon of Finland is used as a 'cutline' object. The result is four small scale hillshading layers of the area of Finland that are used by the new maps for resolution level 2-5 tile matrices.

Setting up the processing of DEMs, for resolution level 6-13 tile matrices, was more complicated, because Elevation model 2 m is used where it is available. The two DEMs have significantly different resolutions, and the values at their borders vary significantly. The first task is to resample Elevation model 10 m to the same grid as is used by Elevation model 2 m. *gdal_warp* is used to cut pieces from the earlier created 10 m DEM virtual raster according to the area division, in addition to a 140 m buffer. At the same time the pieces are resampled to the same grid as is used by Elevation model 2 m. The resulting DEM pieces are then moved to each rendering area's folder. For each area, a virtual raster, sized according to the area, with a 140 m extra buffer, is created from the 2 m DEM set. *gdal_warp* is used to transform each virtual raster to a GeoTIFF file which is then stored into each areas own file.

Because of the accuracy difference between the DEMs, combining the two DEMS directly would result in unnatural looking contour lines and hillshading. The solution was to use map algebra to melt the two DEMs together. First a cost surface is created that had the value zero where the 2 m DEM was not available. 100 meters from the border of the two DEMs, on the side where the Elevation model 2 m is available, the cost surface pixels are given the value 100. The areas in between are given a value corresponding to

the distance from the values given to areas where the Elevation model 2 m is not available by *gdal_warp* in the previous step. The cost surface is created by processing the 2 m DEM pieces with *gdal_proximity.py*. However, *gdal_proximity.py* gives no values to pixels where the distance from the closest target value is more than approximately 93000 meters. The issue was overcome by dividing the 2 m DEM into four smaller pieces and then merging the pieces back together after the cost surface has been calculated.

Gdal_calc is then used to combine the 2 m and resampled 10 m DEM pieces. To produce hillshading the following formula is used:

$$'((2\_m\_DEM * cost\_surface) / 100) + ((10\_m\ DEM\_in\_2\_m\_grid * (100 - cost\_surface)) / 100)'$$

The resulting DEM pieces are then resampled to produce coarser resolution DEMs of the same area with resolutions of: 4 meters / pixel, 8 meters / pixel, 16 meters / pixel, 32 meters / pixel, 64 meters / pixel, 128 meters / pixel. All the DEM pieces are then processed with *gdal_dem* to produce seven hillshading layers for each area. During the process TIFF world files are created for each DEM piece. The darkness of each layer is adjusted with the GIMP curves tool.

Contour lines that are derived from Elevation model 10 m are much smoother than curves derived from the Elevation model 2 m. Smoothing contour lines from the 2 m DEM was necessary to produce a desirable result. A study by Oksanen & Sarjakoski (2005) shows that passing a low pass filter on a DEM before contouring is one way to smooth the contour lines, and may give better results than to smooth the contour lines after they have been generated. For the contour lines, *gdal_filter.py* is used on the 2 m DEM pieces, with a low pass filter matrix depicted in *Figure 122*. *gdal_calc.py* is used with the previously introduced map algebra, replacing the 2 m DEM pieces with the filtered 2 m DEM pieces, to produce DEM pieces that can be used for contour lines. The resulting DEM pieces are then used to derive coarser resolution versions of them, with resolutions of: 4 meters / pixel, 8 meters / pixel, 16 meters / pixel, 32 meters / pixel and 64 meters / pixel. All DEM pieces are then processed with *gdal_contour* to produce all required contour lines. The contour lines are stored by *gdal_contour* into a PostGIS da-

tabase. The final step of the DEM processing removes contour lines that are shorter than a resolution level specific length value. Removing the shortest contour lines, contour lines that would only be a few pixels long, results in a cleaner look for the maps.

0.0625	0.125	0.0625
0.125	0.25	0.125
0.0625	0.125	0.0625

Figure 12. The low pass filter values that was used for the contour lines.

The whole process was automated with a Bash script that first creates the virtual raster layers and cuts the pieces of the Elevation model 2 m and 10 m that are required to create hillshading and contour lines for each rendering area. The Bash script then writes new Bash scripts for each rendering area that will complete the DEM processing separately for each rendering area, for resolution levels 6-13. The DEM pieces and the Bash script are stored in each rendering areas own folder. Another Bash script was written that contains the operations for creating hillshading for zoom levels 2-5. Finally, a single python script was written that executes all DEM processing Bash scripts in each rendering area folder.

5.2.4 Tree Symbol Points

The data processing for the 'Forest map' map type includes generating tree symbol points from the Corine Land Cover 2012 raster. The values of the raster are first reclassified to produce new raster layers, each including only values of a single type of forest, where all other values are removed. These raster layers are then polygonised. Tools for these tasks were searched for. QGIS and PostGIS are both able to polygonise raster data and both are able to generate random points. Both were tested for the purpose. The random point generating algorithm of QGIS resulted in an uneven spread of points within polygons, and was very slow. Even though points were generated randomly, areas of larger polygons are left without any points, while other parts of the polygons may have many points near each other. PostGIS does not have any built-in operation for generating points within polygons. However, such an algorithm was easy to script. The algorithm offered a much more flexible solution.

Points are set to be generated within polygons by using a regularly spaced 40 meter x 40 meter grid. The algorithm goes through the bounding box of the polygon, one grid square at a time, beginning from the top left corner, going left to right and from top to bottom, until reaching the lower right corner (*Figure 133*). The algorithm generates a point in a square and checks whether the point is within the polygon or not. If the point is within the polygon, the point is stored in a PostGIS database. The algorithm then moves on to the next square. If the point is outside the polygon, the algorithm attempts the same procedure two more times. If all three attempts fail to place a point within the polygon, the algorithm moves on to the next square without storing any points. The reason why the algorithm was set to attempt to generate the point three times within the polygon was that small forest areas for example, those consisting of a single 20 meter x 20 meter pixel in the Corine Land Cover 2012 raster, would be more likely to get a point within them and to have a tree symbol representing the forest. For example, a square where a point is attempted to be placed, and that has only a fourth of its surface covered by a polygon representing a forest, leads to an approximately 58% chance for a point to be generated within it.

Points are given an attribute that stores information about within what type of forest the point is located. When all points for a forest type are generated, the table is extended with an additional column where random numbers are generated that are used to distinct between tree symbols that are used. The range of random numbers that are generated depend on the distribution of tree symbols that is used for each forest type. The final part of the process deletes all points that are within water bodies. Water bodies were derived from the NLS PostGIS databases. Points are tested whether they are inside water polygon objects. Any points within water polygons are deleted from the database. All points are stored within a single table regardless of what type of forest the points are located within or what type of tree symbol the point is used for. This enables all points to be organized to be rendered from north to south, enabling overlapping tree symbols to be rendered correctly. All SQL commands that are required for the process were stored into a text file, from where they can be copied and used when there is a need to generate points.

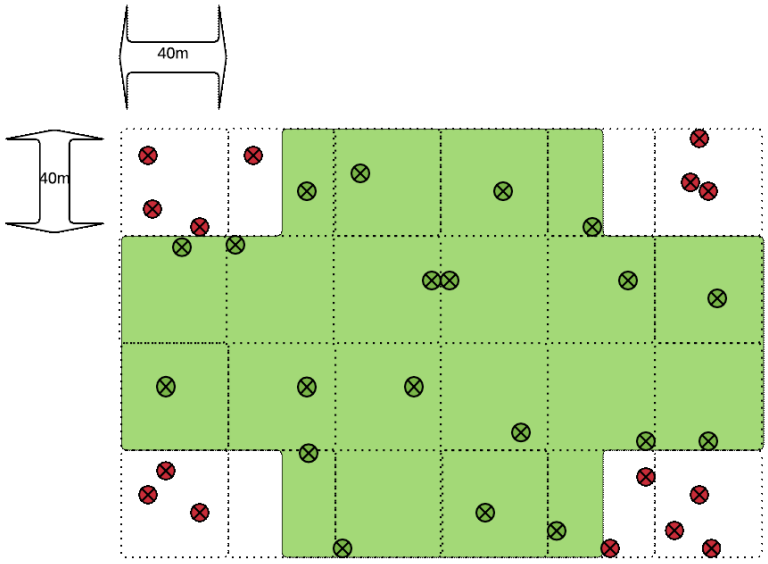

Figure 13. A sketch of how points were generated within the forest areas.

5.2.5 Importing of Superficial Deposits Datasets to PostGIS

The superficial deposits datasets were downloaded from the GTK 'Hakku' service as MapInfo TAB files. A database was created for the datasets in PostGIS. The TAB files were then opened in QGIS from where they were moved with the Database Manager plug-in to PostGIS. This solution requires manual input. However, it is a relatively fast process compared to other data processing in the process. No automated solution was found quickly and due to time constraints, it was not possible to search for software that would allow the process to be automated.

In PostGIS the superficial deposits are extended with an additional text column, where the dataset scale, as given by GTK, is stored. The text in the column is used in the maps as extensions of labels for polygons. This lets users know the scale of the source data that is used for each superficial deposits polygon. This becomes important in higher resolution tile matrices where the highest resolution data is not available for all parts of the map, and a lower resolution datasets are used.

5.3 Map Design

Map design was added to the process by writing tile scheme definitions and properties of tiles to a TileCache configuration file and map style descriptions into MapnikXML stylesheets. The TileCache configuration file and MapnikXML stylesheets are copied to

each rendering area folder, where the coordinates are altered to match the coordinates of the area.

Style descriptions in the MapnikXML stylesheets were not finished for the first version of the map generation process that is presented in this thesis. The style for all features does not match the desired styles for the maps. However, all stylesheets were created, and all required geographic features were included in the stylesheets and given some style description. To finish the styles, the style descriptions for some objects still have to be altered. The style descriptions of each map type were divided into five stylesheets. Each stylesheet includes the style descriptions of one or more resolution levels for a single map type. The resolution levels were selected so that each stylesheet contains mostly data from just one of the NLS small scale databases, or the PostGIS topographic databases. This results in less confusion between data that is used, because the databases contain many tables with the same name. The resolution levels of the stylesheets are 2-5, 6-7, 8, 9-10 and 11-13.

MapnikXML stylesheets were mainly written with a text editor. In the early stages of the development process CartoCSS stylesheets were written in TileMill, with the intention to convert the finished stylesheets to MapnikXML before rendering. However, adding large amount of data to TileMill slowed down the GUI significantly and the style writing process became slow and inefficient. Converting the file format from CartoCSS to MapnikXML with TileMill did not affect the rendering of the outcome file. However, the generated MapnikXML stylesheet was not optimally converted for continued stylesheet editing, for example MapnikXML entities were not generated from CartoCSS entities.

In MapnikXML stylesheets, the root element is called ‘Map’. The ‘Map’ element has two types of child elements, ‘Layer’ and ‘Style’ (*Figure 14*). ‘Layer’ elements have two child elements, ‘StyleName’ and ‘Datasource’. ‘StyleName’ contains information about which ‘Style’ elements the ‘Layer’ element uses. ‘Datasource’ elements contain information about what source data the ‘Layer’ element uses, and how to access it. Style descriptions are contained within ‘Style’ elements which have one or more ‘Rule’ child elements. ‘Rule’ elements have ‘Filter’, and different types of ‘Symbolizer’ child elements, for example ‘PolygonSymbolizer’, ‘LineSymbolizer’ and ‘PointSymbolizer’. ‘Filter’ elements are used for setting conditions for object attributes that have to be ful-

filled by objects in order to be rendered, for example the filter could be set to render objects with a height attribute only when the height would be at least a certain value. The symbolizer elements contain style descriptions, for example a 'LineSymbolizer' element could include information about line width and line color. (Mapnik, 2015)

The order, by which Mapnik renders objects, is determined by the 'painter's algorithm' (Mapnik, 2012). The painter's algorithm renders objects in the order they are written in the MapnikXML stylesheets. 'Layer' elements are rendered in the order they appear in the Stylehseets, using Style elements in the order they are referred to in the Layer elements. Rule elements determine the order within Style elements. Individual objects are rendered in the order they appear in the file or table that they are stored in. Compositing operations enable certain style effects to be achieved that are not normally possible because of restrictions of the painter's algorithm. For example, the compositing operation destination over will render the objects only where there is not any other objects rendered. The compositing operation destination out will delete all data from where the object would otherwise be rendered. By combining these two operations, an object may be rendered to appear to be on top of another object, which is on top of a third object which is on top of the first object. This was utilized in the stylesheets for the new maps, for example to create vignetting for water bodies and text halos that do not cover objects of other color than the color of the text. However, to achieve these effects, some objects are required to be rendered more than once, decreasing tile rendering speed.

```
4136  <Style name="stone">
4137    <!-- Stone -->
4138    <Rule>
4139      &minZoom12;
4140      &maxZoom12;
4141      <PointSymbolizer allow-overlap="yes" file="&SPath1;stone.svg" transform="&SScale12;" />
4142    </Rule>
4143    <Rule>
4144      &minZoom13;
4145      &maxZoom13;
4146      <PointSymbolizer allow-overlap="yes" file="&SPath1;stone.svg" transform="&SScale13;" />
4147    </Rule>
4148  </Style>
4149  <Layer name="stone" status="true" srs="&srs3067;">
4150    <StyleName>stone</StyleName>
4151    <Datasource>
4152      <Parameter name="type">postgis</Parameter>
4153      <Parameter name="connect_timeout">0</Parameter>
4154      <Parameter name="host">localhost</Parameter>
4155      <Parameter name="dbname">database</Parameter>
4156      <Parameter name="user">&user;</Parameter>
4157      <Parameter name="password">&password;</Parameter>
4158      <Parameter name="table">(SELECT geom FROM stone) as foo</Parameter>
4159      <Parameter name="geometry_field">geom</Parameter>
4160      <Parameter name="extent">&extent;</Parameter>
4161    </Datasource>
4162  </Layer>
4163
```

Figure 14. Stylesheet example, 'Layer' and 'Style' elements.

Mapnik can only generate square and dot symbols to represent point objects. Because the maps use a wide variety of different types of symbols for points and lines, SVG and PNG icons were collected, created. The NLS has made the icons used in the NLS topographic map series available in github, and most icons were collected from there. Tree symbols used in the Viherkehä Forest map were available from the FGI. All additional icons were created with GIMP and InkScape, a FOSS vector image manipulation program.

Writing the stylesheets with MapnikXML, meant that tiles had to be viewed with the map viewer when styles were assessed. A small batch of tiles was rendered when changes were made to the stylesheets. The map viewer developed with Leaflet enabled the tiles to be quickly displayed once they had been rendered.

TileCache configurations were added to the Bash script that writes the configuration file and stores it within each rendering area folder. The TileCache configurations are written by adding three types of configurations, cache, layer, and basic configurations. The cache and basic configurations, apply to each layer in the file. For the new maps, the cache configurations include the type of addressing scheme that was desired and the path to the folder where the tiles are to be stored. The basic configurations include the image format to be used. The layer configurations include the rendering engine to be used, the CRS to be used, the size of tiles, the extent of the area to be rendered, the path to the stylesheet that is to be used, and metatiling options. Because there are several stylesheets that are used, a separate layer configuration had to be written for each stylesheet. The extent of each rendering area is different, which is why each rendering area requires its own configuration file.

6 Outcome and Discussion

In this chapter, the outcome of the development process is presented and evaluated and findings from the study are discussed. In Section 6.1, the completed map generation process is presented. In Section 6.2, the qualitative evaluation of the map generation is presented. In Section 6.3, findings and the outcome of the development of the map generation process are discussed.

6.1 The Map Generation Process

After completing the first two stages of the development process, a new model of the map generation process was sketched. This refined model includes all production flow lines for the new maps (*Figure 155*). The map generation process consists of several data processing sub-processes that generate all data for the tile rendering system. The tile rendering system renders tiles according to the tile properties set in TileCache configurations and to the styles that are defined in the MapnikXML stylesheets.

Scripts for executing the data processing sub-processes are kept separate from each other, which enables updating data from a single source without processing all data. Also, tile rendering scripts are kept separate from data processing scripts, to enable rendering of tiles without unnecessary processing of the source data.

Data is processed mostly within PostGIS. However, the DEMs that are processed to generate hillshading and contour lines are processed with GDAL and the produced files are stored in the rendering area folders. Mapnik reads the resulting GeoTIFF hillshading files from these folders. Also the Corine Land Cover 2012 raster layer is read by Mapnik from the source data folder where it is stored. Data processing commands for PostGIS are stored as SQL commands in text files, from where the commands can be copied and used in PostGIS. Manual work is required to input the commands. The processing of the DEMs is automated with Bash scripts and creates all required hillshading layers and contour lines when executing the scripts. One script produces pieces of the DEMs into each rendering area folder. Another script produces the low resolution hillshading of the whole area of Finland, and stores them into the small scale map folder. Stored in each rendering area folder, a third script processes DEM pieces of the areas into hillshading and contour lines.

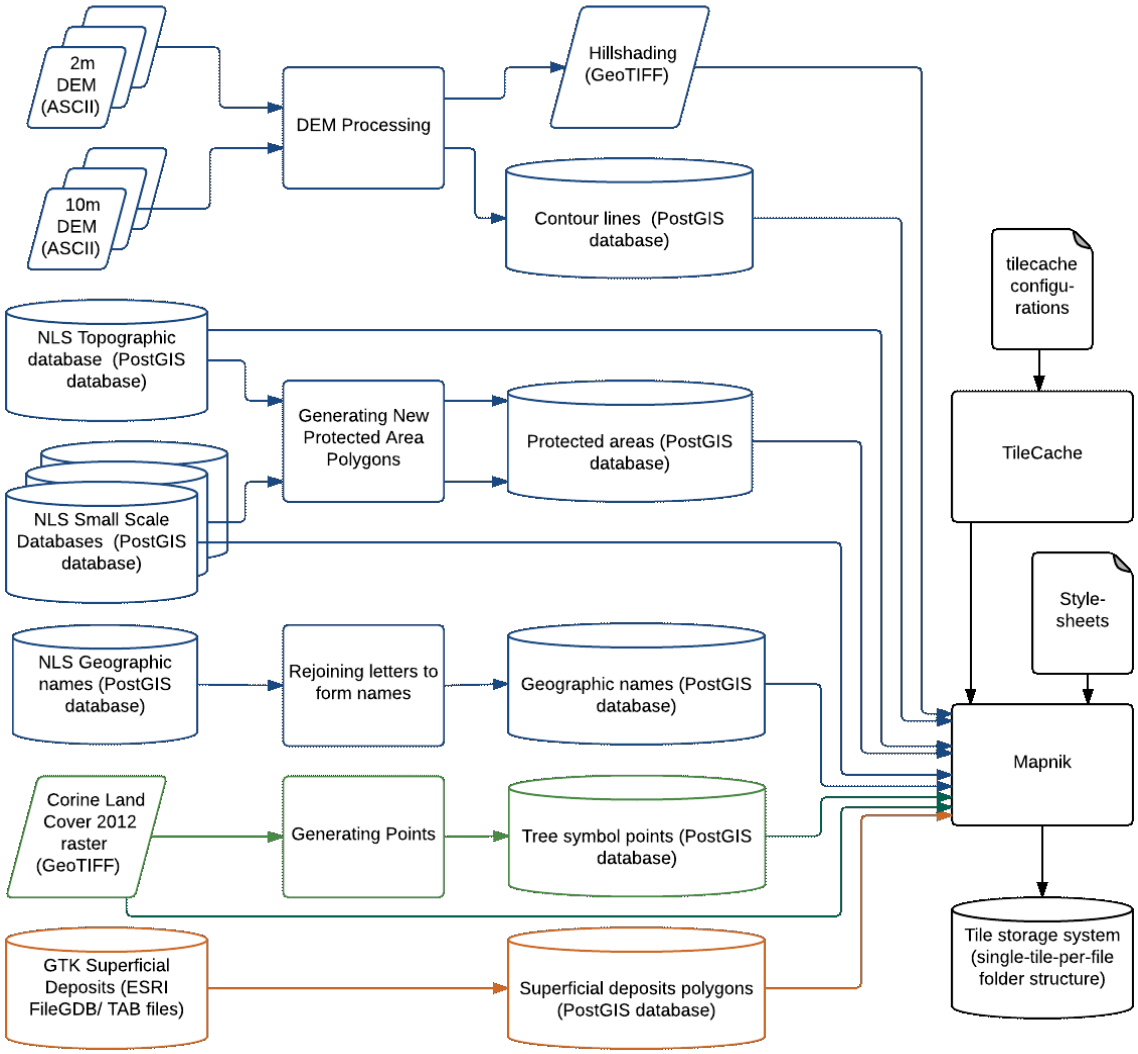

Figure 15. Process model of the finished process.

Tiles are rendered by executing Bash scripts from the command line. Resolution level 2-5 tiles are rendered all at once by the same script. Rendering of tiles at resolution levels 6-13 is divided between the rendering areas. There are 40 rendering areas that have to be rendered to produce a complete map of Finland.

The MapnikXML stylesheets for the new maps were written so that they include all objects that are to be rendered for the maps. However, the map design for the new maps was not finished for this first version of the map generation process. To enable quick updating of map design, a Bash script was written that updates all MapnikXML stylesheets, TileCache configurations and the *GoogleDisk.py* files within rendering area folders at once. The extent of the area to be rendered is edited within each stylesheet and TileCache configuration file by the script, and the GoogleDisk.py that provides the

addressing scheme is also edited also so that tiles are named with their addresses in the JHS 180 grid.

6.2 Qualitative Evaluation

The final stage of the development process was to evaluate the map generation process. The evaluation was carried out qualitatively based on the original requirements set for the process. This assessment was to help determine the efficiency of the solution that was produced and to identify areas that required improvements. The qualitative evaluation can also be used as a base for requirements for future development work.

The map generation process was first evaluated on how well it fulfilled the capabilities desired by the FGI. The map generation process was tested by generating test tile sets from the resolution levels 6-13 and all tile matrices from resolution levels 2-5 (*Figure 16*). The tests showed that tiles are rendered mostly according to the desired configurations. The borders of areas next to each other are also correctly rendered and do not contain cut text or mismatching hillshading or contour lines. However, when using metatiling, additional tiles are rendered to the north and east of the rendering areas. These tiles miss some of the data that is rendered on them when they are produced by the correct rendering area and hence cannot be used. However, tiles that are rendered and stored in the tile storage system are not rendered a second time by Mapnik. Additional blank tiles are not a problem, other than slightly increasing the time it takes to render the maps.

To get an idea of the data processing and rendering tests were conducted with a PC with 23.5 GB of RAM memory, a 3.4 GHz, and a 1.9 TB hard-drive. Although the PC had a 4-core processor, the process was not configured to use more than one for the processing. Tiles of the resolution level 2-5 were rendered in a few minutes. However, rendering tiles for one rendering area (B3), for the ‘Topographic map’, map type took approximately 7 hours. Times were also documented for generating hillshading and contour lines (*Table 8*).

Table 8. DEM processing time examples.

Process:	Area:	Time:
Cutting DEM pieces for area	J1	8 min.
Cutting DEM pieces for area	I2	22 min.
Cutting DEM pieces for area	I3	39 min.
Generating hillshading and contour lines	J1	4 h. 25 min.
Generating hillshading and contour lines	I2	9 h. 4 min.
Generating hillshading and contour lines	I3	14 h. 44 min.

Generating the contour lines from the highest resolution DEM pieces appeared to be the most time consuming task in the DEM processing. It also appears that processing areas that are not completely covered by the DEM, such as area J1, are processed faster. The test cases were only processed and rendered once, and represent a small fraction of the total data processing and tile rendering, and hence, are not reliable for making exact calculations on how long the data processing and map rendering would take. However, the tests show that rendering the tiles on the hardware that the system was built on, without additional optimization of the speed of the process, could take several weeks. The process is required to be developed further to generate maps more efficiently.

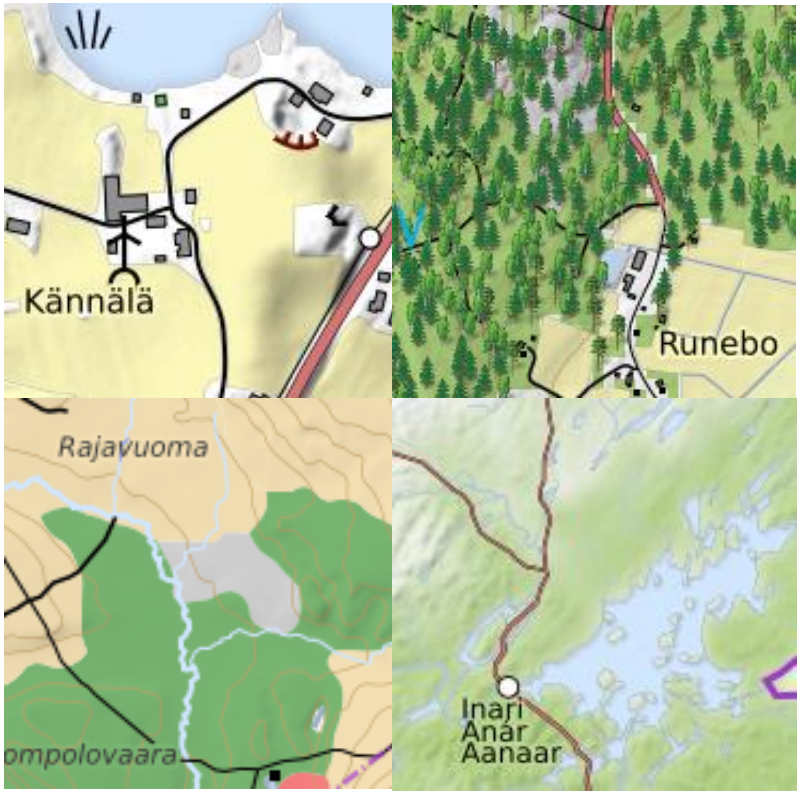

Figure 16. Screenshots of some generated sample map pieces.

The map generation process was not completely automated in the sense that maps would be generated from source data with a single command. However, hillshading and contour lines can be generated with a single command for each area by which the processing is divided. Tiles for each area can be generated in the same way. The rest of the data processing is processed mostly within PostGIS with SQL commands that can be scripted to be executed in order. Writing scripts for these processes and combining all scripts would allow almost the whole process to be automated. The exception is the importing of Superficial Deposits datasets from FileGDB or MapInfo TAB formats to PostGIS, which was done manually from QGIS. Discovering a new method for accomplishing the task could also enable this task to be automated. However, automating the whole process so that it would process all data and render all parts of the maps with a single command is not recommended. For example, it is unnecessary to run the whole map generation process only part of the source data is updated.

Although map style descriptions were not fully completed for the first version of the process, many cartographic capabilities of Mapnik were tested. Compositing operations enable vignetting for water bodies. Mapnik also enables the use of the *multiply* compositing operation. The effect of *multiply* is that bright areas of the hillshading do not affect the source image, but instead the gray parts of the hillshading layer are multiplied to the already rendered parts of the image. Text halos that only cover objects of the same color as the text can be achieved to some degree by Mapnik using the compositing operations. However, with multiple colors of text, depending on the color and rendering order of the rest of the source data, it can be very complicated, or impossible to achieve this effect. For all these effects the use of compositing operations also required the rendering order to be altered and some objects to be rendered more than once. This slowed down the rendering speed of the map generation process. The test batches of tiles that were rendered revealed that, for the resolution levels 6-13, the hillshading causes areas outside of the extent of the DEMs that are part of each area, to be rendered with dark gray. This affects all areas outside the borders of Finland. The dark color is caused by GIMP not reading GDAL noData values. When the brightness of the hillshading is adjusted with GIMP, noData pixels are given the value zero.

The process can be expanded with additional map types by following the same method that was used for the first three maps. It requires adding new data processing operations for new source data, automating the processes, and adding map design to the tile render-

ing system. Existing scripts can be expanded with commands for additional data processing operations, or new scripts may be written. Tile scheme options need to be added to scripts that generate the TileCache configuration files for each rendering area, or the files need to be modified individually. MapnikXML stylesheets need to be written manually, although, when using the same data as for the existing maps, copying the existing stylesheets and editing the copies may speed up the process. Stylesheets then need to be stored within each rendering area folder. The script that copies the stylesheets of the rendering area folders can be expanded to include the new stylesheets.

Based on the qualitative evaluation, the most critical improvements for the process is to finish the map stylesheets for the maps, and to speed up the processing. Fully automating all data processing could speed up the updating of source data, and could be achieved easily, as most of the required SQL commands are stored, and could be copied to scripts that would execute them in the required order. It would be necessary to edit the DEM processing so that the redundant parts of the hillshading raster layers are removed.

6.3 Discussion

The outcome of the development of the map generation process reveals that it is possible to build a map generation process for rendering three multi-scale tiled raster maps with different use contexts by utilizing FOSS tools. The outcome also proves that such a process can be automated for the most part. However, full automation of the map generation process presented in this thesis was not achieved.

Because the area that is rendered for each map is large, it was expected that rendering would be slow. Data processing, most notably, processing high resolution DEMs into hillshading and contour lines takes a significant amount of time. The combined size of the NLS Elevation model 2 m and 10 m source data is more than 520 GB, which is likely to explain why the processing is slow. The process needs to be developed further to make processing of data and rendering of maps more efficient. There are several approaches that could be used to accomplish this. First, the process could use multiple hardware instances that would process data and render tiles simultaneously. Second, optimizing the databases and SQL commands used to retrieve data may speed up the processes. Finally, more efficient algorithms and tools may be discovered for processes if more effort is focused on software comparisons, although testing a wide variety of

tools is time consuming. The map generation process is designed to allow it to be used by multiple hardware instances over a network. The process divides the processing of DEMs, and rendering of tiles for resolution level 6-13 into smaller areas. Multiple hardware could access the files that execute the processing and rendering for each area, and process the data and render the tiles for them, allowing areas to be rendered simultaneously. However, because the system produces additional tiles outside the rendering areas, the system should be developed so that the additional tiles are removed before they are stored in the tile storage system.

The most demanding part of the developing process was the adding of data processing for maps. Data processing is important because data is not always available in a form in which it can be used directly for rendering maps. However, adding data processing took a significantly longer time than building the map rendering system. Because different maps use different source data, it is important to carefully review the data that is available when estimating the length of the development process. The hillshading and contour line generating process was the single most demanding sub-process to develop. Large raster layers are processed slowly, and testing parts of the process causes long waiting times to see results. Testing smaller raster layers first reduces the waiting time, but may not reveal all issues that occur.

Software for tiled map making appears to be widely available. The rapid development of software for map making processes makes it difficult to make software comparisons that are valid for long periods of time. For example, a big factor in the selection of Mapnik as the rendering engine was that Mapnik includes compositing operations. Compositing operations were added to GeoServer in version 2.7 (GeoServer, 2015c), released on March 21st, 2015 (GeoServer, 2015d) and to MapServer in version 7.0, released 24th of July, 2015 (MapServer, 2015e). The new versions of these software now provide the capabilities on which the choice to select Mapnik was based on. Rendering speed is difficult to compare, although recent benchmarking data (MapSurfer.NET, 2015) of several rendering engines show that Mapnik holds a significant advantage in terms of rendering speed to both GeoServer and MapServer. However, benchmarking data that use different data, styles, and tile schemes should always be viewed with caution. Rendering speeds may vary depending on the maps that are being produced. In practice, it is highly inefficient to use time on testing all possible alternatives, and selec-

tions have to be made according to previous user experiences and documentation, even though it may not lead to the optimum solution.

Research on tiled map making appears to be focusing on vector tiles or optimal tile rendering strategies, for example methods that predict what areas to be rendered in advance and what areas to be rendered on-demand. Methods for optimizing the rendering speed could decrease rendering times for high resolution maps with large areas. As tiled maps have the unique feature of being divided into small fragments, rendering them on several separate hardware instances and then storing them in a single tile storage system could potentially enhance the speed of the process.

There appear to be varying opinions on the optimal tile size to be used. According to Sample and Ioup (2010) the most efficient tile size is 512 x 512 pixels. However, the Finnish JHS 180 recommendation, and map services such as Google maps uses tiles of 256 x 256 pixels. The tile size of 256 x 256 pixels can be seen in other contexts too, such as being the default tile size of tiles in several software including TileCache, Leaflet and OpenLayers. Although the optimal tile size may depend on several elements, such as the map client that is used to view the tiles and the resolution of the screen, recommendations such as the JHS 180 could be more specific on why the tile size should be used, and perhaps, if there are situations where using another tile size would be justified.

7 Conclusions

In this thesis the development and outcome of a tiled map generation process were presented. The aim of the thesis was to provide insight on tiled map making, by building an automated map generation process that generates three multi-scale tiled raster maps with varying use context, using FOSS tools.

The development process began by sketching a simple process model, based on which a plan for the development of the map generation process was composed. The development of the map generation process was divided into three stages. In the first stage, a tile rendering system, the central element of the map generation process, was built. The tile rendering system generates the tiles and stores them into a tile storage system. In the second stage, production flow lines were for the three maps were built. The production flow lines include the data processing operations and map design for each map. In the third stage, the map generation process was qualitatively evaluated. The first two stages required software to be selected. When a need for new software components was revealed, the compulsory requirements for the software components were listed, and compatible software was searched for. Alternative software components with significant impact on the final outcome of the map generation process were also compared with eachother before a selection was made between them.

The development process revealed that there are many alternative FOSS solutions for developing a map generation process that renders tiled raster maps. It also revealed that the process can be automated for the most part. Implementing the data processing operations was the most time consuming part of the development process. It is also time consuming to process all source data, especially high resolution raster data, such as the NLS Elevation model 2 m and 10 m. In addition the rendering of tiles for large areas is time consuming. Finding more effective methods to process data and render maps would benefit the process. The process is to be developed further to allow the data processing and tile rendering to be processed on multiple hardware instances. Software for tile based map making is widely available. However, the results of comparing such software expire quickly, and comparisons on rendering speed, that are valid for anything other than the test cases may be difficult to conduct. Research on tiled map generation processes should focus on how the speed of tiled map generation processes that render tiles for large areas could be increased.

References

Antoniou, V., Morley, J. Haklay, M., 2009. Tiled Vectors: A Method for Vector Transmission over the Web. Teoksessa: *Web and Wireless Geographical Information Systems -9th International Symposium, W2GIS 2009, Maynooth, Ireland, December 7-8, 2009. Proceedings.* Springer Berlin Heidelberg, pp. 56-71.

Batty, M., Hudson-Smith, A., Milton, R. Crooks, A., 2010. Map mashups, Web 2.0 and the GIS revolution. Annals of GIS. *Annals of GIS,* vol. 16, no. 1, pp. 1-13.

Braga, V. G., Oliveira, W., Sacramento, V. Cardoso, K. V., 2014. *Descriptive Modeling of the Web Mapping.* São Paulo, GeoInfo 2014 - Brazilian Symposium on Geoinformatics. [Online] Available at: http://www.geoinfo.info/proceedings_geoinfo2014.split/Paper07-F-p17.pdf [Accessed 23 8 2015]

Corcoran, P., Mooney, P., Bertolotto, M. Winstanley, A., 2011. View- and Scale-Based Progressive Transmission of Vector Data. Teoksessa: *Computational Science and Its Applications - ICCSA 2011 - International Conference, Santander, Spain, June 20-23, 2011. Proceedings, Part II.* Springer Berlin Heidelberg, pp. 51-62.

Dufilie, A. Grinstein, G., 2014. Feathered Tiles with Uniform Payload Size for Progressive Transmission of Vector Data. *Web and Wireless Geographical Information Systems.* Seoul: Springer Berlin Heidelberg, pp. 19-35.

Eldrandly, K. Soad, N., 2013. A Knowledge-Based System for GIS Software. *The International Arab Journal of Information Technology,* vol. 10, no.2.

freegis.org, 2015. *FreeGIS Database.* [Online] Available at: http://freegis.org/database/?cat=0&_ZopeId=51052300A7KXKYcLvnc [Accessed 02 09 2015].

Gaffuri, J., 2012. Toward web mapping with vector data. Teoksessa: N. Xiao, M. Kwan, M. *Geographic Information Science -7th International Conference, GIScience 2012, Columbus, OH, USA, September 18-21, 2012. Proceedings.* Springer Berlin Heidelberg, pp. 87-101.

García, R. de Castrom. J.P. Verdú, E. Verdú, M.J. Regueras, L.M., 2012. Web Map Tile Services for Spatial Data Infrastructures: Mangement and Optimization. *Earth and Planetary Sciences, Cartography - A Tool for Spatial Analysis.* InTech.

GDAL, 2015a. *GDAL - Geospatial Data Abstraction Library.* [Online] Available at: http://www.gdal.org/ [Accessed 8 21 2015].

GDAL, 2015b. *Software using GDAL.* [Online] Available at: http://trac.osgeo.org/gdal/wiki/SoftwareUsingGdal [Accessed 21 8 2015].

GeoServer, 2015a. *GeoServer is an open source server for sharing geospatial data.* [Online] Available at: http://geoserver.org/ [Accessed 15 9 2015].

GeoServer, 2015b. *GeoServer User Manual.* [Online]
Available at: http://docs.geoserver.org/2.6.2/user/ [Accessed 2 10 2015].

GeoServer, 2015c. *GeoServer 2.7.0.* [Online]
Available at: http://geoserver.org/release/2.7.0/ [Accessed 2 10 2015].

GeoServer, 2015d. *GeoServer 2.7 released.* [Online]
Available at: http://blog.geoserver.org/2015/03/23/geoserver-2-7-released/
[Accessed 2 10 2015].

GTK, 2013. *Maaperä 1:1000 000.* [Online]
Available at: http://tupa.gtk.fi/paikkatieto/meta/maapera_1m.html
[Accessed 9 23 2015].

GTK, 2015a. *Superficial deposits 1:20 000 / 1:50 000.* [Online]
Available at: http://tupa.gtk.fi/paikkatieto/meta/maapera_20_50k.html
[Accessed 23 9 2015].

GTK, 2015b. *Superficial deposits 1:100 000.* [Online]
Available at: http://tupa.gtk.fi/paikkatieto/meta/maapera_100k.html
[Accessed 23 9 2015].

GTK, 2015c. *Superficial deposits of Finland 1:200 000 (sediment polygons).* [Online]
Available at: http://tupa.gtk.fi/paikkatieto/meta/maapera_200k.html
[Accessed 23 9 2015].

Haklay, M., Singleton, A. & Parker, C., 2008. Web Mapping 2.0: The Neogeography of the GeoWeb. *Geography Compass,* vol. 2. no. 6, pp. 2011-2039.

Hardy, P., Briat, M.-O., Eicher, C. Kressman, T., 2004. Database-Driven Cartographic from a Digital Landscape Model with Multiple Representations and Human Overrides, Leicester, UK: ICA Workshop on 'Generalisation and Multiple Representation.

JUHTA, 2013. *JHS 180 Paikkatiedon sisältöpalvelut - Liite 1 Karttakuvapalvelu,* Julkisen hallinnon tietohallinnon neuvottelukunta.

Kettunen, P., Sarjakoski, L. T., Ylirisku, S. Sarjakoski, T., 2012. Web Map Design for a Multipublishing Environment Based on Open APIs. *Online Maps with APIs and WebServices* Springer Berlin Heidelberg, pp. 177-193.

Longley, P. A., Goodchild, M. F., Maguire, D. J. Rhind, D. W., 2015. *Geographic Information Science and Systems.* 4th ed. John Wiley & Sons, Inc. p. 477.

MapBox, 2015. *Compositing operations.* [Online]
Available at: https://www.mapbox.com/tilemill/docs/guides/comp-op/
[Accessed 10 10 2015].

MapBox, 2015. *TileMill.* [Online]
Available at: https://www.mapbox.com/tilemill/ [Accessed 31 8 2015]

Mapnik, 2012. *MapDesign.* [Online]
Available at: https://github.com/mapnik/mapnik/wiki/MapDesign [Accessed 31 8 2015]

Mapnik, 2015. *XMLConfigReference.* [Online]
Available at: https://github.com/mapnik/mapnik/wiki/XMLConfigReference [Accessed 31 8 2015]

MapServer, 2014. *SLD.* [Online]
Available at: http://mapserver.org/ogc/sld.html#sld [Accessed 15 9 2015].

MapServer, 2015a. *Welcome to MapServer.* [Online]
Available at: http://mapserver.org/index.html [Accessed 15 9 2015].

MapServer, 2015b. *Image Formats.* [Online]
Available at: http://mapserver.org/mapcache/formats.html [Accessed 15 9 2015].

MapServer, 2015c. *Cache Types.* [Online]
Available at: http://mapserver.org/mapcache/caches.html [Accessed 15 9 2015].

MapServer, 2015d. *MapServer OpenLayers Viewer.* [Online]
Available at: http://mapserver.org/cgi/openlayers.html [Accessed 15 9 2015].

MapServer, 2015e. *Version 7.0.0 Announcement.* [Online]
Available at: http://mapserver.org/development/announce/7-0.html#announce-7-0 [Accessed 2 10 2015].

MapSurfer.NET, 2015. *Benchmarking Mapping Toolkits in Tile Seeding.* [Online]
Available at: http://mapsurfernet.com/blog/benchmarking-mapping-toolkits-in-tile-seeding [Accessed 2 10 2015].

Neteler, M., Bowman, H. M., Landa, M. & Metz, M., 2012. GRASS GIS: A multi-purpose open source GIS. *Environmental Modelling & Software,* vol. 31, pp. 124-130.

Niemelä, O., 2004. *Maasto ja Kartta - Kartanvalmistajan ja-käyttäjän käsikirja.* Keuruu: Genimap Oy Otavan kirjapaino. p. 175.

NLS, 2015a. *Maastotietokanta.* [Online]
Available at: http://www.maanmittauslaitos.fi/digituotteet/maastotietokanta [Accessed 25 09 2015].

NLS, 2015e. *Digital Elevation Model 2 m Availability.* [Online]
Available at:http://www.maanmittauslaitos.fi/sites/default/files/km2_tuotantotilanne.pdf [Accessed 10 10 2015].

NLS, 2015b. *Karttalehtijako TM35.* [Online]
Available at: http://www.maanmittauslaitos.fi/digituotteet/maastotietokanta [Accessed 23 9 2015].

NLS, 2015c. *Korkeusmalli 10 m.* [Online]
Available at: http://www.maanmittauslaitos.fi/digituotteet/korkeusmalli-10-m
[Accessed 23 9 2015].

NLS, 2015d. *Korkeusmalli 2 m.* [Online]
Available at: http://www.maanmittauslaitos.fi/digituotteet/korkeusmalli-2-m
[Accessed 23 9 2015].

OGC, 2010. *Web Map Tile Service Implementation Standard,* [Online] Open Geospatial Consortium Inc. Available at: http://portal.opengeospatial.org/files/?artifact_id=35326 [Accessed: 20.8.2015]

Oksanen, J. Sarjakoski, T., 2005. *The EVRS and the need for contour updating in national topographic maps.* [Online] Available at: http://cartesia.org/geodoc/icc2005/pdf/poster/TEMA3/JUHA%20OKSANEN.pdf

Oksanen, J., Schwarzbach, F., Sarjakoski, L. T. Sarjakoski, T., 2011. Map Design for a Multi-Publishing Framework – Case MenoMaps in Nuuksio National Park. *The Cartographic Journal,* vol. 48, no. 2, pp. 116-123.

PostGIS, 2015. *About PostGIS.* [Online]
Available at: http://postgis.net/ [Accessed 23 9 2015].

PostgreSQL, 2015. *About.* [Online]
Available at: http://www.postgresql.org/about/ [Accessed 9 23 2015].

Proj.4, 2015. *Proj.4.* [Online]
Available at: https://trac.osgeo.org/proj/ [Accessed 12 09 2015].

Quinn, S. Gahegan, M., 2010. A Predictive Model for Frequently Viewed Tiles in a Web Map. *Transactions in GIS,* vol. 14, no. 2, pp. 193-216.

Robinson, A. H., Sale, R. D., Morrison, J. L. & Muehrcke, P. C., 1984. Elements of cartography. 5 ed. United States of America: John Wiley & Sons, Inc. p. 544.

Roth, R. Donohue, R. Sack, C. Wallace, T. Buckingham, T.., 2014. A Process for Keeping Pace with Evolving Web Mapping Technologies. *Cartographic Perspectives,* Issue 78, pp. 25-52.

Sample, J. T. Ioup, E., 2010. Tile-Based Geospatial Information Systems: Principles and Practices. Springer Science + Business Media. p. 237.

Schmidt, M. Weiser, P., 2012. Web Mapping Services: Development and Trends. *Online Maps with APIs and WebServices.* Springer Berlin Heidelberg, pp. 13-21.

Sipilä, T., 2015. *Web Map Service Architecture for Topographic Data in Finland.* Rio de Janeiro, Brazil, International Cartographic Conference - Maps Connecting the World. Available at: http://icaci.org/files/documents/ICC_proceedings/ICC2015/papers/6/fullpaper/T6-440_1428670244.pdf [Accessed 25 9 2015]

SpatiaLite, 2015. *SpatiaLite.* [Online]
Available at: https://www.gaia-gis.it/fossil/libspatialite/home [Accessed 23 9 2015].

SQLite, 2015. *About SQLite.* [Online]
Available at: https://www.sqlite.org/about.html [Accessed 23 9 2015].

Steiniger, S. Hunter, A. J., 2013. The 2012 free and open source GIS software map – A guide to facilitate research, development, and adoption. *Computers, Environment and Urban Systems,* Issue: 39, pp. 136-150.

SYKE, 2014. *Corine Maapeite.* [Online] Available at:
http://metatieto.ymparisto.fi:8080/geoportal/catalog/search/resource/details.page?uuid={D54C552F-F7F7-489B-8B1E-E093D93C7386} [Accessed 23 9 2015].

TileCache, 2015. *TileCache -- Web Map Tile Caching.* [Online]
Available at: http://tilecache.org/#tilecacheconfig [Accessed 23 8 2015].

TileStache, 2015. *TileStache API.* [Online]
Available at: http://tilestache.org/doc/ [Accessed 23 8 2015].

Tsou, M-H., 2005. *Recent developments in Internet GIS.* [Online]
Available at: http://www.gisdevelopment.net/technology/gis/techgis_002pf.htm [Accessed 25 08 2015].

Tsou, M-H., 2011. Revisiting web cartography in the United States: The rise of user-centered design.. *Cartography and Geographic Information Science,* vol. 38, no. 3, pp. 250-257.

Tsou, M.-H. & Smith, J., 2011. *Free and Open Source Software for GIS education.* [White paper] Available at: http://geoinfo.sdsu.edu/hightech/WhitePaper/tsou_free-GIS-for-educators-whitepaper.pdf [Accessed 12 09 2015].

Tyner, J. A., 2010. *Principles of Map Design.* The Guildford Press. p. 259.

Virrantaus, K., Fairbairn, D. Kraak, M.-J., 2009. ICA Research Agenda on Cartography and GIScience. *Cartography and Geographic Information Science,* vol. 36, no. 2, pp. 209-222.